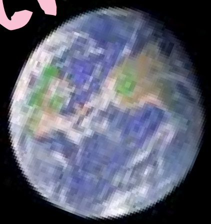
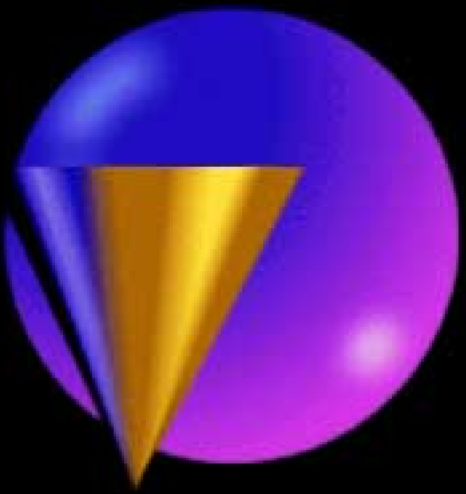
The Magazine for C/S and Internet-Programming 01/2000

US$ 25.00

SDT international

Software Development Techniques





Support

VO meets Flagship

Owner Drawn Menus

Database: Compound Storage

Xbase++: WEB Application Adapter II

INHALTSVERZEICHNIS

INHALTSVERZEICHNIS

Impressum

Editors and Publisher:
VOCA GmbH
Grossviehbergstr. 48A
91213 Hersbruck
Germany
Tel.: +49-9151-8181-41
Fax: +49-151-8181-42

Http://www.vocager.de

Authors 1/2000:
Jan V. Balek
Dieter Crispien
Uwe Holz
Georg S. Lorrig
Ralf Saborowski
Rod da Silva
Dieter Stelzner
Ivo Wessel
Erik Wynn
Michael Zech

Titlepage:
C. Leithold
N. Pfleger
[WAM Dortmund]

Dear Reader

For this issue of "Software Development Techniques", we have reworked the layout of the magazine to fit changing times. We hope that you will find SDT easier read because of these changes. At this point, I would like to thank our GUI Pope, Ivo Wessel, who, through long and heart-felt discussions, convinced us of the need for some changes.

Keyword Changes:
Clandestinely, quietly, service releases of CA-Visual Objects are still appearing. While other world-shaking companies of every size and color sing an anthem with every minor product adjustment, Computer Associates goes their own way. A quick message in their new WEB-Forum announced the availability of the 2.5a1 patch, which can be downloaded from ftp://ftp.cai.com/pub/vo.

Keyword Web-Forum:
Controversially, Computer Associates cut their association with CompuServe last year. Many of our readers suddenly lost their most important source of information and support.. Computer Associates reacted to the situation and setup a peer-to-peer solution. Aware users threw themselves on this new forum, which is quite easily reached from the Internet, and the culture shock began.

Say what one will about CompuServe, its organization and accessibility were extremely good when compared with this somewhat exotic solution, which Computer Associates has created for us. VOCA attacked this problem immediately, and as of December had begun development of the CAYMan OffLine Reader. The first release now lies behind us, and VOCA quite excited about how the users are responding to it.
This issue's CD includes a free 30-day trial version. With Computer Associates-World and C.A.R.E. approaching, we wanted to report our new SDT concept and the development of CAYMan.

Finally, heart-felt congratulations to Mr. Jürgen Goschke, as the winner of the trip to CA-World.

Please, enjoy reading your SDT,
Meinhard Schnoor-Matriciani
[VOCA GmbH, Germany]

Ralf Saborowski

OLE Structured Storage or “How to keep all DBFs and indexes of an application within one file”

OLE Structured Storage with CA-Visual Objects was not given much attention until now. It is a very good example of native COM programming with CA-Visual Objects and it opens quite interesting programming solutions. This article explains the concept behind OLE Structured Storage and has an example of how to ‘twist’ the CA-Visual Objects RDDs to make them store database files in OLE Compound Files. A Compound File Viewer that enables to view the structure of OLE Compound Files will be presented.

There are actually two motives behind this article. On one hand I wanted to be able to keep all DBF and index files of an application in one large file since the time I was a Clipper developer, thus for more then ten years. The mere sight of a directory with dozens of DBF and NTX files (composite indexes were not available with Clipper Summer 87) felt like cold shower running down my back. This accumulation of files somehow appeared more vulnerable to me than a single file. First it was possible to overlook a missing file during application backup or transfer. Second end users could easily open and manipulate files with tools like dBase and even corrupt data.

The concept of OLE Structured Storage exists since the introduction of OLE 2.0 in Windows 3.1. What hides behind this term? One of the central characteristics of OLE 2.0 is the ability to embed documents in other documents. For example Excel table can be embedded in a Word document. The Excel table in turn can contain further documents like sound files or a complete PowerPoint presentation.

This kind of complex document (also called Compound Document) organization required a new persistent storage mechanism. You can hardly expect the user to keep track of all individual embedded document components to safely transfer the document. We only deal with embedded documents here, you still have to do this with the so-called Linked documents. The goal was to keep all components of a complex document in a single file and leave the format of the embedded files intact whenever possible. This is how the idea of OLE Compound Files was born. Compound files are files, which in principle can again contain a complete hierarchical file system.

OLE Compound files are already used in CA-Visual Objects. The project catalog %CavoDir%\Projects\Project.vo is an OLE Compound File, which can be easily proven with the Compound File Viewer that will be introduced later in this article. My personal interest in the concept of OLE Structured Storage and my long-term desire to place DBFs and index files in a single file were the incentives to solve this problem at once. And then, Elvis was nagging me, well, you know.

Working with Compound files

The functions to create and manipulate Compound files are contained in dynamic link libraries Ole2.dll and Storage.dll. Functions to create and open a Compound File are StgCreateDocfile() and StgOpenstorage(). Both functions expect a set of parameters including the name of the Compound File. A typical call to create a new Compound File looks as follows:

```
StgCreateDocFile(;
 PTR(_CAST,; Multi2Wide(cStorage)),;
 _OR(STGM_READWRITE,STGM_SHARE_EXCLUSIVE),;
 0,;
 @pStg)
```

The first parameter is the name of the file to be created. Note that this parameter has to be passed not as a character string or PSZ but as a pointer to a wide character (Unicode) string. This applies to all file names used with Compound Files. Runtime functions Multi2Wide() and Wide2Multi() in the system library provide conversion between wide character (Unicode) and single character strings. The second parameter specifies the mode, in which the created file has to be opened. In this case we open it for exclusive read and write access. Compound files can also be opened for shared access and in a so-called transaction mode. The last two modes are hardly recommended with the current implementation of the Storage.dll. The Compound File specification allows shared file access, however with the current implementation it is only possible to open components of a Compound File exclusively, even if the Compound File itself is opened in shared mode. So in reality a Compound File opened in shared mode will only permit several users to have simultaneous exclusive access to different components within the file.

If a Compound File is opened for shared access, the transaction mode can be indicated in addition. It allows to roll back to the initial state after a set of operations. Since contents of the Compound File are copied to a TEMP directory during transactions and there are no mechanisms for conflict recovery during a commit, this mode has little practical importance.

The third parameter of the StgCreateDocFILE() call is reserved and set to a NULL_POINTER. The fourth parameter finally takes us closer to the true nature of the storage API. It is passed by reference and receives a COM interface for the created storage object if the function call was successful (return value is S_OK = 0). All further access to the compound file has to be done through this COM interface.

The function to open an existing compound file looks similar:

```
StgOpenStorage(  Multi2Wide(cStorage),;
  NULL_PTR, _OR(STGM_READWRITE,STGM_SHARE_EXCLUSIVE),;
  NULL, 0, @pStg)
```

The first parameter is again the file name as a wide character string. The second parameter points to a previous opening of a root storage object and can be passed in addition to the file name. The third parameter is the desired access mode, the fourth parameter permits to filter out certain elements within the storage and the fifth parameter is reserved. Again a COM interface is returned by reference through the sixth parameter.

Interfaces, interfaces, interfaces...

You should be already familiar with the basics of COM programming, so I don’t have to deal with it here. A detailed paper about native COM programming with CA-Visual Objects 2.x can be found in [1].

Both StgCreateDocFile() and StgOpenstorage() return so-called IStorage interfaces. These represents the ‘main directory’ of the Compound File. However with Compound Files storage objects are used instead of directories. A root directory is accordingly called a root storage object. Files in a compound file are called stream objects or streams. The IStorage is defined in CA-Visual Objects as follows:

```
CLASS IStorage INHERIT MyIUnknown
  DECLARE METHOD CreateStream
  DECLARE METHOD OpenStream
  DECLARE METHOD CreateStorage
  DECLARE METHOD OpenStorage
  DECLARE METHOD CopyTo
  DECLARE METHOD MoveElementTo
  DECLARE METHOD Commit
  DECLARE METHOD Revert
  DECLARE METHOD EnumElements
  DECLARE METHOD DestroyElement
  DECLARE METHOD RenameElement
  DECLARE METHOD SetElementTimes
  DECLARE METHOD SetClass
  DECLARE METHOD SetStateBits
  DECLARE METHOD Stat_
```

You can easily recognize methods for creating, deleting and renaming of storage objects and streams. In detail these methods fulfill the following tasks:

IStorage:Createstream()
Creates and opens a new stream to which data can be written like to a normal file. After a successful execution Createstream() returns an IStream interface.

IStorage:Openstream()
Opens an existing stream. As mentioned before streams can only be opened exclusively. An attempt to open a stream for shared access will return an error code. A successful execution of Openstream() returns an IStream interface.

IStorage:Createstorage()
Creates and opens a new storage object nested within this storage object. This procedure is somewhat similar to the creation of a subdirectory in a file system. Successful execution of Createstream() returns a new IStorage interface.

IStorage:Openstorage()
Opens an existing nested storage object. Like streams nested storage objects can be opened only exclusively. A successful Openstream() execution returns an IStorage interface.

IStorage:CopyTo()
Copies the entire contents of an open storage object including streams and nested storage objects to another storage object.

IStorage:MoveElementTo
Copies or moves a nested storage object or stream from this storage objects to another storage object.

IStorage:Commit()
Ensures that changes made to a storage object opened in the transaction mode are permanent.

IStorage:Revert()
Discards changes made to a storage object opened in the transaction mode.

IStorage:EnumElements()
Enumerates all streams and nested storage objects of a storage object. An IEnumSTATSTG interface is returned after successful execution.

IStorage:DestroyElement()
Removes the specified storage object or stream from this storage object. A storage object can be successfully deleted only if it does not contain any more streams or nested storage objects.

IStorage:RenameElement()
Renames a stream or a nested storage object.

IStorage:SetElementTimes()
Allows setting of the time values for the creation, last access, and last change of a stream or a nested storage object.

IStorage:SetClass()
Assigns the specified CLSID to the storage object.

IStorage:SetStateBits()
This method is documented as reserved and should not be used for the time being.

IStorage:Stat_()
Provides status information (name, date, etc...) for the current storage object.
As already mentioned, access to streams is obtained through the IStream interface, which inherits not from IUnknown direct, but from ISequentialStream.

```
CLASS ISequentialStream INHERIT MyIUnknown
  DECLARE METHOD Read
  DECLARE METHOD Write

CLASS IStream INHERIT ISequentialStream
  DECLARE METHOD Seek
  DECLARE METHOD SetSize
  DECLARE METHOD CopyTo
  DECLARE METHOD Commit
  DECLARE METHOD Revert
  DECLARE METHOD LockRegion
  DECLARE METHOD UnlockRegion
  DECLARE METHOD Stat_
  DECLARE METHOD Clone
```

These methods support operations necessary for handling streams (files), as for example read, write and search.

IStream:Read()
Is used to read data from a stream.

IStream:Write()
Writes data to a stream.

IStream:Seek()
Streams have a seek pointer just like files. This function changes the seek pointer position so subsequent reads and writes can take place at a different locations in the stream.

IStream:SetSize()
Changes the size of the stream object.

IStream:CopyTo()
Copies a specified number of bytes from the current seek pointer in the stream to the current seek pointer in another stream.

IStream:Commit()
Ensures that changes made to a stream open in transaction mode are permanent.

IStream:Revert()
Discards changes made to a stream opened in the transaction mode.

IStream:LockRegion()
Restricts access to a specified range within a stream. Since the current implementation of Compound files supports only exclusive access to streams, this function has no practical value.

IStream:UnlockRegion()
Removes a lock set with IStream:LockRegion(). Since the current implementation of Compound files supports only exclusive access to streams, this function is likewise irrelevant.

IStream:Stat_()
Returns status information (name, date, etc..) for the current stream.

IStream:Clone()
Creates a new stream object with its own seek pointer that references the same bytes as the original stream.
Another interface you will need to work with Compound files is IEnumSTATSTG. It is returned by Istorage:EnumElements() and is used to run through all nested storage objects and streams:

```
CLASS IEnumSTATSTG INHERIT MyIUnknown
  DECLARE METHOD NEXT_
  DECLARE METHOD Skip
  DECLARE METHOD Reset
  DECLARE METHOD Clone
```

An IEnumSTATSTG interface represents an enumeration of all streams and nested storage objects. Its pointer indicates the current item. When the enumeration is created the pointer is set on the first item in the collection.

IEnumSTATSTG:Next()
Returns the specified number of streams or nested storage objects in the enumeration. The requested information is returned in STATSTG structures. Keep in mind that the storage object or stream name is returned as a wide character string (Unicode) and it is the callers responsibility to explicitly allocate and free memory for this string by using CoTaskMemFree().

IEnumSTATSTG:Skip()
Skips one or more items in the enumeration.

IEnumSTATSTG:Reset()
Resets the enumeration sequence to the first item.

IEnumSTATSTG:Clone()
Creates another IEnumSTATSTG interface that is an exact copy of the first.

In practice

All interface definitions including method declarations are included in the library you can import from the SDT-CD as storage.aef. Armed with this information we can write the first program that uses Structured File Storage:

```
FUNCTION Start()
LOCAL cb AS DWORD
LOCAL c AS STRING
LOCAL pStg AS IStorage
LOCAL pStm AS IStream

// Create new Compound File
StgCreateDocFile( PTR(_CAST,Multi2Wide(“test.dfl”)),;
    _OR(STGM_CREATE,STGM_READWRITE,;
    STGM_SHARE_EXCLUSIVE),0, @pStg)
// Create new stream
pStg:CreateStream(Multi2Wide(“test.txt”),;
  _OR(STGM_CREATE,STGM_READWRITE,;
    STGM_SHARE_EXCLUSIVE),0, 0, @pStm)
  // Write to stream
pStm:Write(PTR(_CAST, “Hello World!”), 13,@cb)
// Close stream
pStm:Release()

// Close Compound File
pStg:Release()

// Open Compound File
StgOpenStorage( Multi2Wide(“test.dfl”),NULL_PTR,;
  _OR(STGM_READ,STGM_SHARE_EXCLUSIVE),NULL,0,@pStg)

pStg:OpenStream(Multi2Wide(“test.txt”),0,
_OR(STGM_READ, STGM_SHARE_EXCLUSIVE),0, @pStm)
// Read from stream
c:=Buffer(100)
pStm:Read(PTR(_CAST, c), 100, @cb)
c := Left(c, cb-1)
? c
pStm:Release()
pStg:Release()
WAIT
```

To keep the size of the code to minimum it comes without error handling. The program creates a compound file with the name ‘test.dfl’ (the extension DFL stands for document file). It then creates a stream named “Test.txt” in the root storage and writes the text “Hello World! “ into the stream. After the stream and storage are closed they are opened in the read only mode and the text from the stream is read and displayed.

Repeated Istorage:Createstream() and Istorage:Createstorage() calls allow to create as many storage objects and streams as desired including complex hierarchies within compound files. We are getting closer to our goal to store all DBF and index files of an application in a single file.

Poor Man’s Doc View

How can we know that the file built in the preceding example really has the intended structure? We cannot look inside of compound files, like it is possible with directories in the Windows

Explorer. Originally I wanted to refer to a tool named DocView.exe, which comes with Microsoft Visual C++. This tool can open compound files and display nested storage objects and streams in a Tree View. Stream contents can be viewed in hexadecimal representation. Then I thought that after all good developers make their own tools and the result was the "Poor man's Doc View.AEF" you can find on the companion CD. This application is a pretty good Compound File viewer. You can use it to view the file built in the previous example as well as many interesting files, for example Word documents, Excel tables and CA-Visual Objects MDF files. Simple Microsoft Office documents contain only few streams in the root storage, however once you add some macros and perhaps embed another OLE object it starts to look a lot more interesting (see illustration 1). Embedded objects like an Excel table are stored in the Object Pool storage in more or less their original form.

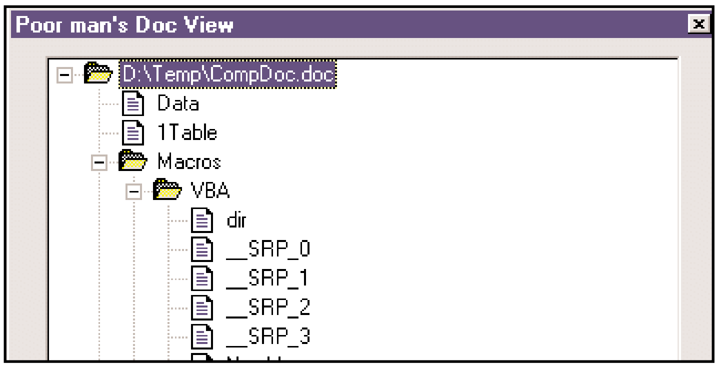


Fig. 1: A Word document with macros and inserted Excel table in the Doc file viewer

To view the contents of a stream in a Hex editor double-click on a stream icon. Caution is required, because the window, which opens, is a Hex editor and all changes in the window will be written back into the stream without confirmation request. This way a Word document can be corrupted and made useless in an instance.

You can rename storage objects and streams in the viewer via single click of the left mouse button and delete whole branches of directories with the Del key. No request to confirm deletion will follow so caution is required!

Compound RDD storage

The problem that we naturally face is that to replace all file operations by IStorage and Istream calls, we need the RDD source code. Strictly speaking that is not much of a problem for me, but nevertheless I wanted to develop a solution, which works with existing RDDs.

I wasn't involved with the details of the RDD system for a long time, however I still remembered that all file operations are performed via functions of the runtime library (CAVORT20.DLL). I assumed that the runtime functions are made available to the RDD through an array of function pointers similar to a Vtable. In that case file operations could be easily rerouted by replacing the relevant function pointers with pointers to own routines that make calls to the IStorage/IStream interfaces. Unfortunately a look at the relevant macros indicated that instead of using an array of function pointers, RDD uses direct runtime calls. These are in essence hard-wired once the program is loaded into the memory.

Runtime functions used in the RDD system for file operations, are based Win32 API calls like CreateFile(), ReadFile(), WriteFile(). This led to the idea to replace the Win32 API functions in runtime RDD with own routines. The result would be that not only RDD function calls, but also FCreate(), FOpen() and even the VOERROR.LOG would be rerouted into a single compound file.

Replacing of imported functions like I described it above, is an easy exercise for programmers familiar with design of linkers. When the application is loaded Windows gets hold of one fixed address for each import function in the import table. All calls to a runtime function within a module (an EXE or DLL) are passed indirectly through these fixed import addresses. Replacing addresses in the import table with addresses of own routines would reroute runtime function calls to own routines. Of course, these new routines should have same calling semantics as the original functions. I call this technology subclassing of DLLs. Indirect calls to DLL routines can be easily traced down with the GoVest debugger [2] in the single step mode.

The file "Storage Rdd.aef" on the companion CD contains a library, with all necessary logic for subclassing of the runtime DLL and some routines that allow to reroute file read/write operations to a compound file. The following Win32 API functions are replaced:

CloseHandle()	CreateDirectory()
CreateFile()	DeleteFile()
FlushFileBuffers()	LockFile()
ReadFile()	RemoveDirectory()
SetCurrentDirectory()	SetFilePointer()
UnlockFile()	WriteFile()

It is important that these functions are only replaced for the Runtime DLL. Calls made to these functions from the application (EXE) or from other DLLs should still go through the Win32 API.

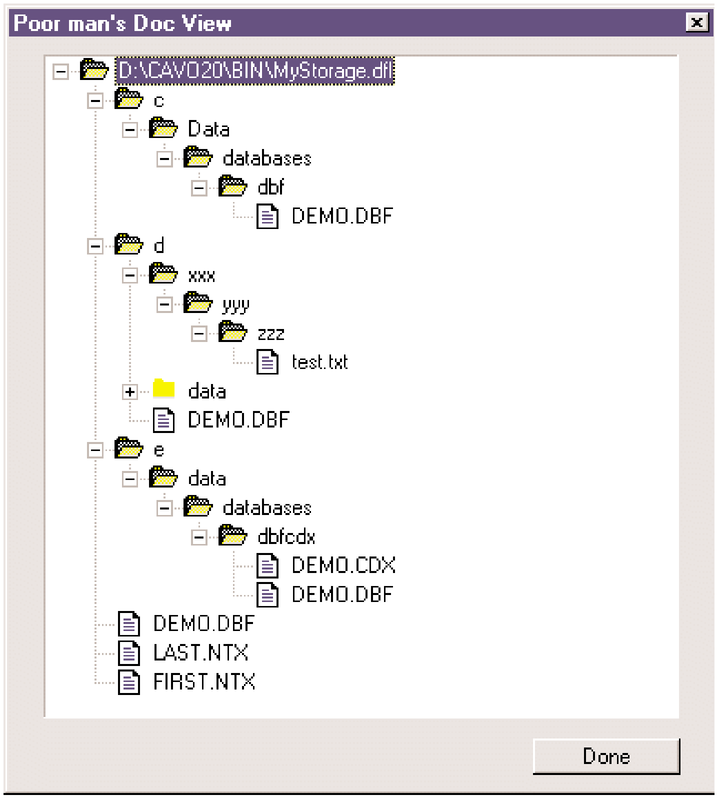
Before you continue reading, import the "Compound DBF test.AEF" from the companion CD. Build and run the application and view the produced file Mystorage.dfl in the Compound File Viewer. Then take a look at the start function of the test program. The start function consists mostly of classic Clipper style code to create and index DBF files. The actual creation of the DBF files takes place in the function C_DEMO() that creates a DBF file with 100 records in a typical Clipper manner. Indexes using DBFNTX or DBFCDX are built in the Start() function itself.

The program does not generate DBF and index files as files in the file system, as you would expect from this code. Instead compound file MyStorage.dfl is created. Once you examine it in the Compound Document File Viewer you will find that DBF and index files are saved as streams within a single compound file (see illustration 2).

The detour is handled by an instance of the class StorageRDD:

```
o: =  storageRDD{ }
```

Once it is initialized all functions listed above will reroute their input/output to a single compound file named Mystorage.dfl.



```
D:\CAVO20\BIN\MyStorage.dfl
  c
    Data
      databases
        dbf
          DEMO.DBF
  d
    xxx
      yyy
        zzz
          test.txt
    data
    DEMO.DBF
  e
    data
      databases
        dbfcdx
          DEMO.CDX
          DEMO.DBF
  DEMO.DBF
  LAST.NTX
  FIRST.NTX
```

Abb. 2: MyDocFile.dbf in Poor Man's Doc View

The name of the compound file can be passed to the StorageRDD class Init() method as the second parameter. The StorageRDD object is system-wide, only one StorageRDD object can exist at a time. An attempt to create a new object will automatically destroy the existing one.

The logic for the subclassing of the runtime DLL is in the Init() method of the StorageRDD class. It can be easily modified to subclass any other DLL or other functions. This way - by sub classing - you can write your own tracers and API replacement routines in CA-Visual Objects.

The StorageRDD class provides the possibility to create nested storages by subclassing the directory management functions. Runtime functions DirMake(), DirChange() and DirRemove() will work with storages if the StorageRDD class is active. If the directory names include drive letters StorageRDD will create sub storages with respective drive letters in the root storage.

Keep in mind that files can be stored as streams only in the current storage (selected with DirChange() and all files (streams) in the current storage have to be closed, before a new storage is selected with DirChange(). New storages can be created without any changes in the current storage. Files can be opened only in exclusive mode. All these limitations are in essence due to the fact that the current Structured Storage implementation permits only exclusive access to nested storages and streams.

The StorageRDD class can be suspended/restored by setting the export variable fActive:

```
o:fActive := FALSE
```

Files already opened will be handled independently the fActive setting. This flag makes a difference only for file creation and directory operations.

Happy End

This article gave you basic tools for using compound files in your own applications. The declarations in the storage.aef contain the necessary classes and the presented examples illustrate how to handle compound files.

How practical this kind of file management really is remains to be seen, considering the limitations in the current version of the storage.dll. Compound files can be a good solution for storing data in proprietary format for exclusive access. The presented StorageRDD class is meant as an interesting example of the possibilities CA-Visual Objects offers as a language, rather then a serious practical application.

Ralf Saborowski
VP R&D
Computer Associates International, Inc.
E-Mail: Ralf.Saborowski@cai.com

Literature:
[1] SDT 01/99
Ralf Saborowski
Native COM with CCA-Visual Objects
[2] SDT 03/99
Ansgar Trimborn
GoVest!

Jan V. Balek

eMail with FlagShip

Everyone wants the ability to send individual and group eMail directly from the existing database programs.

The standard xBase language, which FlagShip also uses, is mainly intended for programming database applications. A typical address database usually contains all of the information, needed to automate sending eMail. Nothing stands in the way.

When the application itself becomes eMail-enabled, the various elaborate detours involving normal eMail programs can be forgotten. Additionally, each addressee receives a separate eMail instead of the unpleasant and impersonal list of CC's (carbon copies) or BCC (blind CC) normally used.

Such an eMail option increases the value of the enabled application.

Prerequisites

Unix (or Linux) provides the necessary eMail dispatcher in the form of the *Sendmail* program. The Unix installation process normally installs *Sendmail* automatically. *Sendmail* can operate on eMail either as a background daemon, or be called directly.

An Internet connection is practically another prerequisite these days. *Sendmail* uses the same log (usually SMTP) as Netscape or most other eMail programs. If such a program is running, no further adminstrative actions are required. If not, the system installation program (such as Yast for SuSE Linux) installs the necessities in just a few minutes.

Of course, properly proper formatting the data sent to *Sendmail* is also required. This manual work is possible, if complex, and usually requires inside system knowledge. The FlagShip library includes the *WebSendMail()* function, in order to make this work as simple as possible, as will be shortly seen. It belongs to a complete family of *Web*()* functions, which are intended for communication with the Internet. These functions are contained in the FlagShip distribution (see [1]), and the source code is available in case modifications are needed (see [2]).

A simple eMail program

The first example represents a simple stand-alone program for dispatching boilerplate to one or more addressees. The name of the text file, as well as the target addresses, are passed through the command line, when the program is called.

The program is quite simple. The only modification, which you must implement, is the entry of your own eMail address in the *#define MY_EMAIL* line at the top of the program. This information goes to the recipient in the *From:* entry and should be corrected. Otherwise, the reciever cannot properly reply.

WebMailDomain() (see [2]) checks if the target address is valid. The return value is either a properly formatted SMTP eMail address (such as XYZ@Domain.XYZ), or empty under error conditions.

WebSendMail() is the core of the program (see [2]), and supports either the *Sendmail* program, or daemons. Before dispatching eMail, it checks the most important data and returns an appropriate return code, for example, 0 for success. In the event

```
// mail1.prg: example for using FlagShip to
//        create and send e-mails
// This example sends an available ascii
//   text file (passed as 1st parameter)
//   to max ten recipiens (1..10) given in
//   parameter 2 to 11.
//
// Compile: FlagShip mail1.prg -o mail1
// Execute: ./mail1 email_text “to-address”
/                 [“to-address” ...]
//    e.g.: ./mail1 body.txt “John Smith
//  <jsmith@nowhere.com>” enyone@domain.net

// change this to YOUR e-mail address!
#define MY_EMAIL “myself@mydomain.de”

// get command-line parameters, check
// plausibility
//
parameters txt_file, p1, p2, p3, p4, p5,;
  p6, p7, p8, p9, p10
local nCount, cEmail, ok

if pcount() < 2
  ? “syntax: “, execName(), “email_text_file” +;
    “‘to-address’ “ +    “[‘to-address’ ...]”
  ?
  quit
endif

if empty(txt_file) .or. !file(txt_file)
  ? “sorry, e-mail text file ‘“ + ;
     txt_file + “‘ not available.”
  ?
  quit
endif

// send the e-mail body to all given
// addressees, display success/error
//
? “Sending an e-mail text ‘“ + txt_file + ;
  “‘ (with bcc to “ + MY_EMAIL + “)”

// process all email adresses
for nCount := 2 to pcount()
  // process command-line parameters
  cEmail := &(“p” + ltrim(nCount -1))
  if empty(cEmail) .or. ;
    empty( WebMailDomain(cEmail) ) // check
    ? “* e-mail address ‘“ + cEmail + “‘ ;
      incorrect, not sent to”
    loop
  endif
  ok := WebSendMail(MY_EMAIL, ;  // from
           “test e-mail”, ;      // subject
           memoread(txt_file) + ;
             chr(13)+chr(10), ; // body
           cEmail, ;             // to..
           NIL, ;                // no cc
           MY_EMAIL)             // bcc
  if ok == 0
    ? “- to “ + cEmail
  else
    ? “* could not sent to ‘“ + cEmail + ;
      “‘, error code =”, ltrim(ok)
  endif
next
?
quit

// input not required, disable curses
//
function cursesinit
return NIL
```

of an error, the return codes are described in detail by the FlagShip manual (see [3]).

The field *Subject:* was feed a literal, in this case. It could also become a command line parameter, or be extracted from the text file.

A note about the command line: If the parameter contains a space, or the characters < >, this parameter must be in single or double quotes.

A Practical Example

The next example represents a somewhat more complex, and more generic, program, which can be inserted with minor modification directly into your application.

For reasons of space, the complete listing could not appear here. However, you will find it on the enclosed CD in the *\flagship* directory, with compilation notes, and additional comments. Since you would probably rather copy this file, rather than type the listing in, let’s devoute our attention to more interesting details.

```
// mail2.prg: example for using FlagShip to
//      create and send e-mails.
// This example is interactive, i.e. it
//     lets you create the common
//     text (or use an asci file) and sends
//     it to to all recipients given
//     in a file or database.

// *** change the following e-mail header
//      data for YOUR domain and company !!
static MY_EMAIL := {
  ‘“my name” <myself@mydomain.de>’,; //from
  “Our Company Name”, ;    // organization
  “myself@mydomain.de”, ;  // send reply to
  “postmaster@myddomain.de”} // return path
```

This time no defines for the sender’s eMail address. Instead, this example uses the entire eMail header structure, which is then passed to *WebSendMail*(). Note: you still need to modify the hard-coded *From:* entry.

```
parameters par1, par2
:: local ....
local aDbfStru := ;
   {{“email”,”c”,60,0}, ;
    {“name”, “c”,30,0}, ;
    {“first”,”c”,20,0}}
// you may redefine the following field
// positions according to YOUR used
// database if any
#define FLD_EMAIL 1
#define FLD_NAME  2
#define FLD_FIRST 3
#define CRLF      chr(13)+chr(10)
```

In order to simplify the modifications you will need to make, field numbers were used instead of field names. This way, you can simply modify the defines to match your database.

Perhaps you caught that Clipper would not allow the order of the declaration statements above, such as the *Local* declaration after an executable statement or the *Parameters* declaration in this case. It is, however, allowed and correct for FlagShip. The later *Local* (or *Static*) declaration increases in some cases the readability, and enables hiding the dynamically scoped variables, either with a later instruction, or conditionally.

```
:: get and check parameters
:: display help on request
:: prompt for file name if not given,
:: create database from text if required

if !empty(cDbfName)
  use (cDbfName) NEW
endif
ok := used() .and. !eof()

// check if at least one valid
// e-mail address is available in
// the database (e-mail field)
if ok
  ok := .F.
  while !ok .and. !eof()
    ok := !empty( ;
      WebMailDomain(fieldget(FLD_EMAIL)) )
    skip
  enddo
  go top
endif
if !ok
  wait “sorry, no address data “ + “available ...”
  quit
endif

:: read the e-mail body from file
:: or create it on-line if not avail.
:: prompt for subject

// send all the e-mails
//
go top
?
while !eof()
  cEmail := WebMailDomain( fieldget(FLD_EMAIL))
  if !empty(cEmail)
    :: format the title
      // if the normailzed e-mail differs
      // from the given one, send both
      // to be sure to be delivered
      cCc := alltrim(fieldget(FLD_EMAIL))
      if upper(cEmail) == upper(cCc)
        cCc := NIL
      endif
      ok := WebSendMail( ;
          MY_EMAIL, ;   // from address
          cSubject , ;  // subject
          cHead + CRLF + CRLF + ;
            cBody + CRLF, ;  // body
          cEmail, ;          // addressee
          cCc)                // carbon copy
      if ok == 0
        nSent++
        ?? “ - sent”
      else
        nFailed++
        ?? “- FAILED”
      endif
    endif
  else
    ? fieldget(FLD_EMAIL), cHead, ;
      “... NOT sent, wrong address”
    nFailed++
  endif
  skip
enddo
wait “done: “+ ltrim(nSent)+ “ sent, “ + ;
     ltrim(nFailed) + “ failed...”
quit

//———————————————————
// Read ascii text data into tempor. dbf
//
```

```
function readToDbf(file, aDbfStru)
:: see source code
return tmpdbf

** eof **
```

Some Tips

If you have recently installed the system, it is worthwhile to test the validity of the web and eMail configurations with, for example, Netscape.

Don't try the patience of friends. Many sites, such as www.hotmail.com, are offering free web-based eMail accounts, which can be activated and ready for testing within seconds. If everything works, the free account can be dropped just as quikkly.

Although network, modem, and ISDN settings usually install without problems, the excellent Linux documentation covering these topics is worthwhile reading. The files can be found under */usr/doc/howto/**, and begin with ISP, PPP, and NET-3. The ZLess program is also an excellent viewer for these documents.

```
zless *ISP*gz
```

If your computer suddenly develops a mind of its own, and starts independantly connecting to the Internet, this is not a computer virus (which are mostly unknown in the Unix world, anyway). It is far more likely to be *Sendmail*, or the *cron* daemon running in background, in an attempt to send unsent eMail. Depending upon your configuration setting in */var/spool/mqueue/var/mailq*, or */var/spool/postfix*, outgoing eMail is only deleted after being successfully sent. The *mailq* command can display the list of unsent eMail. With it, you can check, or even delete, these unsent eMails, directly, with the permission levels *Root* or *Su*. Deleting will stop this undesirable behaviour.

5. References

[1] In this issue a free, fully executable, version of *FlagShip* (for the SuSE 6.2ff or Redhat 6.0ff and similar Linux distributions) is contained on the enclosed CD under *\flagship*. Please see the ReadMe and license.html files.

[2] After installation, source for the standard *Web*()* functions is contained in */usr/FSsrc /tools/webtools*.

[3] The complete FlagShip on-line manual is included with each distribution. A printed manual is also available. See also *man fsman* for further details.

[4] "Daemon" does not mean a type of devil ☺, rather a utility program operating in the background. The abbreviation comes from "Disk And Execution MONitor". See also http://foldoc.ic.ac.uk.

Jan V. Balek
eMail: jvb@multisoft.de

Uwe Holz

CGI APPLICATIONS for Linux/UNIX

More and more developers of Internet applications are confronted by their customers with the demand for a Linux version. That is, since Linux is a modified version of UNIX, this is the sturdiest platform for Internet servers, not to mention performance advantages. This article shows how you can transport the source code of your VO CGI application 1:1 into UNIX. It describes which tools are needed and how an appropriate environment is installed. As a result of this study, special FlagShip versions of the VO Base Classes HTTPCgiContext and LogFile have been developed that makes this port possible.

In the beginning there was... UNIX

Hardly anybody who seriously deals with application development nowadays can avoid the topic of the Internet. It is really amazing to see how rapidly the demand for Web-applications has risen. They are a main topic of this magazine from its first issue on. However, so far mainly Windows/NT was regarded as the general platform for it. The reality looks slightly different:

The Internet itself was in earlier times simply a domain of UNIX, a long time before it became interesting for Windows programmers, not to say, attainable. Still today the UNIX server still forms the crucial components in the Web, and surely for good reason. All standard TCP/IP systems existed already many years before the first NT version. Not very many Internet Providers are deeply opposed to the idea of introducing NT as an alternative platform to the proven UNIX machines. They surely have good reasons for it.

As long as it concerns pure client applications, Windows NT/9x/2000 is of course the first choice. Here Microsoft controls the market hands down. With Internet server applications on the other hand, one does simply not go past UNIX. Lately, Linux has become very popular as the alternate UNIX platform, which additionally confirms the trend.

Ask your Provider

As application developers you should be prepared for this situation. Imagine the following situation: You have developed an Internet application for a customer, he is content, you naturally all the more, pure joy. One day he calls and tells you that he has changed his Provider. The new one, of course, only has Linux servers. “No problem, is it?”, he said without bad alterior motives. At the latest now you become aware of the term platform dependence. Whether your customer really brings you into difficulties with this will still show up. For the time being, we take the described situation as a basis for further considerations.

CGI or what?

First we need to discuss what is required for the porting of a VO application. At first there would be the technology. Just to tell you in advance: everything other than CGI is an illusion. In previous contributions to this topic it has already been referred to this. Also the search for a suitable compiler is useless. Already to the weddings of Clipper there was the FlagShip compiler of Multisoft. At that time it was often dismissed under the term UNIX derivative of Clipper. I think the time for this compiler has now come to life.

The compatibility to Clipper is extremely high. This contribution will show whether and to what extent is suitable for the Porting of VO source text.

The environment

After the questions about the compiler and about the technology are settled, the environment is interesting. For a UNIX beginner like me the new platform is actually already a large hurdle. Apart from the compiler and operating system, also all other tools, e.g. the editor, require a large expenditure of time for training. Who starts back from this can make do with a hybrid environment quite well for the moment.

For this you need a Windows workstation (Windows 9x/NT/2000) and a Linux server. Both are connected directly by TCP/IP or within a LAN.

Linux and FlagShip installation

A special version of FlagShip must be installed on the server. In my case it is the FlagShip release 4.48.7452 for Linux ELF & Glibc-2.1. It is, among other things, also contained on the accompanying CD. My Linux installation is “Linux 2.2.12-32”, a RedHat-6.0-Distribution. Other versions are naturally also possible. A further important package is Samba, a collection of freely available software tools that make it possible to use Linux as file and printer servers for Windows workstations.

Samba is contained in the installation with all usual Linux distributions as an option. Do you happen to know the quickes NT system for file and printer servers? There exists a persistent rumor that this is Linux+Samba.

From my own experience I can confirm that this has a very good performance. A direct comparison is a good topic for a further article, I think. But back to the topic:

Samba recognizes the SMB record on the Linux server. This is used by all Windows versions for common access of data and printers. Thus disc drives and listings like those of NT servers can be mapped on the work station. After the release of such listings the following is to be considered:

Windows98, WindowsNT (starting from service pack 3) and Windows 2000 use coded passwords for access to network disc drives. Therefore the Samba configuration file *smb.conf* has been configured accordingly:

```
[global]
. . .
encrypt passwords = yes
security = user
. . .
```

In addition it is unfortunately necessary to install a special password file on the Linux system. You find the instructions for it in the Samba documentation and/or in supplemental literature.

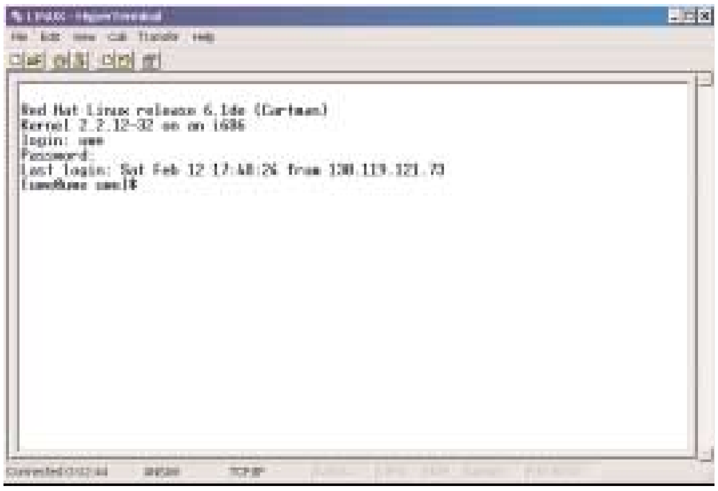
Work station

With the release of listings on the Linux server a large hurdle is overcome: the editor. Now you can use your favorite DOS and/or Windows editor and get along completely without *Vi* (editor that is available on each UNIX). By the way, recently I heard an interesting assessment to it: “Vi is rather difficult to learn. At first sight, it looks just as antiquated as e*dlin* from DOS. But someone who can control it, can also can cook eggs

with it.“ I became curious and tried *Vi*, and I must say, the man seems to know what he is talking about.

Editors are something like a religion for programmers, Vi supporters must therefore forgive me for these remarks. In my next life I will surely work at *Vi*, *Emacs* or another one of the numerous partially graphical UNIX editors a bit more, but for now, I just have no time for that experience...

After the access to the source text is clarified, the compiler remains intact. This one must naturally be operated on the Linux server. But also it is there for an access reason: terminal programs. They make it possible to practically remote control the server from the work station. The hyperterminal of Windows (accessory/communication) is such a program. Via the port 23 a Telnet connection to the server can be developed



```
Red Hat Linux release 6.1de (Cartman)
Kernel 2.2.12-32 on an i686
login: [illegible]
Password:
Last login: Sat Feb 12 17:48:26 from 130.119.121.73
[illegible]$
```

Fig.1: Telnet connection to the Linux server

If this one is perfect, then the Linux server can be served on the level of the command line. Mind you, in a window on the Windows station, thus in the user’s environment.

Compiler and other aids

During training for FlagShip, I was reminded inevitably of my old Clipper days: no GUI, compiler and linker options, (r)Make utility, and batch files. These are called “shell scripts“ in UNIX. However, there are substantially more options so that the individual projects can be configured precisely. All necessary information is in the good and clearly arranged documentation for FlagShip.

The FlagShip compiler itself produces portable C-code which can be compiled with the GNU compiler ggc. This C code by Flagship is not only suited for GNU compilers, but they have the advantage that they are a lot more “intelligent” then their corresponding peers (C compilers by Motorola, Siemens, Sun, etc.).

After the production of this intermediate code the C-compiler itself is called. Fortunately it can process DOS files, thus text files, which use CRLF (0x0D0A) as line changes. UNIX systems normally use only a LF (0x0A). Most DOS or Windows editors have problems with the UNIX format. But for this described environment, editing under DOS/Windows, compiling and linking under Linux, this difference is thus insignificant.

However the Utility make of Linux does not “like“ DOS files. If you work on the Make control file under Windows, this must be converted on the Linux side with the utility program DOS2unix.

In this connection I would like to point out one fault of my popular editor *Edt.exe*: It stores each changed file in capital letters. For Linux this is equivalent to a renaming if the file name did not already consist only of capital letters before (remember that UNIX is case-sensitive). Thus here we must intervene correctly before processing on the Linux server. That can quite simply be dealt with via a so-called Shell script:

```
#
# Prepare Makefile:
# 1. Rename MAKEFILE to Makefile
# 2. Convert from Dos to Unix format
#
mv MAKEFILE Makefile
dos2unix -a Makefile

#
# Rename all involved files from
# uppercase to lowercase
#
mv TEST.PRG     test.prg
. . .

#
# Call make utility
#
make install

#
# Convert Makefile back to Dos-Format for
# your favored Editor
#
unix2dos -a Makefile
```

Contents of the file *Makefile* are described later, first I would like to deal with the porting of existing VO classes.

VO classes for FlagShip

The CGI beginning is as well known based on the fact that the web-server makes parameters and other options available for the respective request via environment variables. The appropriate script process can query these variables and, in the case of a so-called POST Request, read in additional data over the standard input (*stdin*). The result is published as an HTML entity via the standard output (*stdout*). The smallest CGI program looks in FlagShip exactly as in Clipper:

```
? “Content-Type: text/html”
?
?
? “<html>”
? “Hello World!<p>”
? </html>”
?
?
```

The allocating of the handles for in- and output devices as under NT is not even necessary here. So, appropriate classes are to be realized still more easily than in VO. But how concretely does it look with the conversion of VO Source-code?

Compatibility

The simplest beginning is of course just to translate the existing sources with FlagShip. That is exactly how I began. The result was at first discouraging: pages and pages of error messages.

With exact analysis most of them turned out to be trivial. First it strikes that FlagShip uses other variable types than VO, CHARACTER instead of STRING and INTVAR instead of INT, NUMERIC instead of FLOAT. With appropriate #defines that can easily be regulated. These and other pre-settings landed gradually in a header file, which was for obvious reasons I called missing.fh. Here also, all important FlagShip header files are merged so that each individual source file actually only needs missing.fh:

```
#ifndef _MISSING_DEFINED
#define _MISSING_DEFINED

#include “fspreset.fh”
#include “stdclass.fh”
#include “error.fh”
#include “fileio.fh”
#include “stdfunct.fh”
#include “directry.fh”
#include “rddsys.fh”

#define CRLF         CHR(13) + CHR(10)
#define LF           CHR(10)
#define BR           “<br>”
#define CR           CHR(13)

#define E_EXCEPTION 5333

#define INT          INTV
#define STRING       CHAR
#define FLOAT        NUMERIC

. . .

#endif
```

It is true that with the help of this header file the compiler was successful, but during the link phase, however, a set of functions was missed, which mostly had to be rebuilt. Here is a summary of these “subsequently delivered“ functions, the source text is naturally present on the accompanying CD. First the FlagShip sources (*missing.prg*):

Note: The functions above are missed only after the port of existing VO sources. At first, *FlagShip* is Clipper-compatible concerning these functions. I put *FlagShip* to an unusual test. The compatibility concerning VO is scheduled for the next graphical version, which I did not yet have at hand.

Some functions have even been realized in C (cmissing.c) also:

Crypt()	DirChange()
ErrString()	Pow()

One of the three C-interfaces is nearly identical to the one of Clipper. Since *FlagShip* produces C-code itself we have the possibility to embed C source code into the .PRG Files. Here an example:

```
FUNC Rand() AS INTV
    LOCAL_LONG  nRet
    LOCAL       iRet  AS INTV

    #Cinline
        {
        nret = lrand48();
        }
    #endCinline

    iRet := nRet
    RETURN iRet
```

FlagShip presents itself thereby as a very open system, so that all necessary functions can be copied easily.

Thus for functionality, VO source can generally be ported problem-free. Even the OOP syntax is completely supported. Particularly interesting is the fact that also the VO class dbServer exists, this makes it, for example, possible to rebuild certain missing functions:

```
FUNC DBInfo  (n)   AS USUAL
    LOCAL oDBServer AS Object

    oDBServer := DBObject()

    RETURN oDBServer:Info(n)
```

Skin module and initializations

After these preparations and using the functions developed in the course of the work, one can start to act. The basic structure of a CGI-application could be transferred in its essential structure from VO. Additionally two small routines are necessary according to FlagShip documentation, which make important initializations:

ProgramInit()   CursesInit()

Shown below is the complete basic structure of a CGI-application, which essentially corresponds to the VO-CGI-example votest.exe. It also considers the use of the FlagShip Webkit, a special Add-on by Multisoft for the organization of CGI applications. Besides, flagship supports special Web*() functions which were not used here.

```
#include “missing.fh”
#include “httpcont.fh”
#include “httpcgi.fh”
#include “logfile.fh”
// Comment out next line to use webkit
// #include “webkit.fh”

#ifndef _WEBKIT_CH
FUNC ProgramInit()
    CALL fgsUse4html
    RETURN NIL

FUNC CursesInit()
    RETURN NIL
#endif

STATIC FUNC __MyError   (oErr)
    LOCAL cRepl      AS STRING
    LOCAL oError     AS Error
    LOCAL i           AS INT

    oError := oErr
    i := 3
    DO WHILE ProcLine(i) > 0
        cRepl += CRLF
        cRepl += “MODUL “ + ProcName(i)
        cRepl += “, LINE “ + ;
                 NTrim(ProcLine(i))
        i++
    ENDDO
    IF SLen(oError:Description) > 0
        oError:Description += cRepl
    ELSE
        oError:Description := cRepl
    ENDIF
    BREAK oError

CLASS StdCgi INHERIT HTTPCgi
    PROTO METHOD HTTPResponse CLASS StdCgi

#ifdef _WEBKIT_CH
FUNC webkitmain (aEnv AS ARRAY,;
                 aForm AS ARRAY,;
                 oState AS USUAL)
#else
FUNC Start
#endif

    LOCAL oCGI      AS StdCgi
    LOCAL cHtml     AS STRING
    LOCAL oOldError AS USUAL
    LOCAL oError    AS USUAL
    LOCAL oLogFile  AS LogFile
    InitApp()

#ifdef _WEBKIT_CH
    oCGI := StdCgi{aEnv, aForm}
#else
    oCGI := StdCgi{}
#endif
    oOldError := ;
      ErrorBlock({|oErr| __MyError(oErr)})

    BEGIN SEQUENCE
        cHtml := oCGI:HTTPResponse()
        IF SLen(cHtml) > 0
            //
            //  Standard Behavior,;
            //      return a HTML
            //
            oCGI:Write(cHtml)
        ENDIF
    RECOVER USING oError
        cHtml := ;
            oCGI:HTTPErrorMessage(oError)
        oCGI:Write(cHtml)
        oLogFile := LogFile{NIL,NIL,.F.}
        oLogFile:DumpError(oError)
    END SEQUENCE
    ErrorBlock( oOldError )
    oCGI:Close()
    RETURN
```

The program basics which are common to all CGI-applications contains a few FlagShip characteristics:

Each class can be defined by a prototype, similar to C++, thus the compiler will presuppose this class as available. In addition, it is now able to call the methods of the respective class directly (early binding). This concept works here very similarly as in VO. However, here the agreement of the prototype of the class and the respective methods already is enough:

```
CLASS StdCgi INHERIT HTTPCgi
    PROTO METHOD HTTPResponse CLASS StdCgi
```

The actual respective code can be written out into a separate module. In connection with this example, this makes absolute sense because the actual application is realized here exactly.

The function InitApp() (module missimg.prg) is responsible for special FlagShip pre-settings, as the example shows:

```
FUNC InitApp() AS USUAL
    //
    //  All special intialization goes here
    //
    LOCAL_LONG  nTemp

    nTemp := Seconds() + ;
             Day(Date()) + SecondsCPU()
    #Cinline
        //
        // init value for Rand()
        //
        srand48(ntemp);
    #endCinline

    // Set lowercase for all file names
    FS_SET(“lower”, .T.)
    FS_SET(“pathlower”, .T.)
    RETURN NIL
```

The call of the C-runtime-function srand48() sets a special initial value for the initialization of pseudo-random numbers by means of lrand48(). The two FS_SET() calls guarantee that all file and path names are converted into lowercase letters before their use. All UNIX systems are case sensitive concerning file names. That means that, for example, the file Test.PRG does not have anything to do with test.prg, since it concerns two different filenames with regards to case sensitivity.

Looks like VO, doesn’t it?

The classes themselves can be moved over nearly 1:1 with VO. Only Win32-API calls must be replaced adequately. That concerns above all the determination of the full file name of the application and some functions to work with the Registry. The latter are rarely deleted. The name of the application can be determined with the call of a FlagShip runtime function:

```
ExecName(.T.)
```

With the assignment of class names it must be referred here to a FlagShip characteristic. Similarly, as in Clipper, only the first 10 characters are significant with keywords. This also applies, among other things, to the names of classes. Consider therefore the following references:

- All classes must differ clearly from each other by the first 10 characters in the name.
- As long as a class is not used as a base class for the further transmission, it can possess a name that is longer than 10 characters.
- The name of a base class must not be longer than 10 characters, otherwise a compiler error arises.

From (1) –(3) it follows that, with the assumption of the classes from VO to FlagShip, the class names had to be changed as follows:

```
HTTPContext -> HTTPCont
HTTPCgiContext -> HTTPCgi
```

Since the entire source code is present to the individual classes on the accompanying CD, I would like to deal only with the prototypes.

Class log file

The class *LogFile* was brought over particularly for compatibility reasons. It shows a beginning to the logging of errors and other information. An alternative to it offers the already mentioned Ad-on *Webkit* with many possibilities.

LogFile provides the appropriate file according to standards in the listing of a CGI-application. It receives the name of the CGI-application and the extension log:

```
PROTO CLASS LogFile
    PROTECT cFName       AS STRING
    PROTECT cBackupName AS STRING
    PROTECT cDLLPath     AS STRING
    PROTECT lFileLog     AS LOGIC
    PROTECT lAppend      AS LOGIC

    PROTO METHOD Init         CLASS LogFile
    PROTO METHOD DumpError    CLASS LogFile
    PROTO METHOD DebugMsg     CLASS LogFile
    PROTO ACCESS ServerPath  CLASS LogFile
    PROTO ACCESS FName        CLASS LogFile
    PROTO ASSIGN FName(c)     CLASS LogFile
    PROTO ACCESS Append       CLASS LogFile
    PROTO ASSIGN Append(l)    CLASS LogFile
    PROTO ACCESS Log2File     CLASS LogFile
    PROTO ASSIGN Log2File(l) CLASS LogFile
```

Classes HTTPCont and HTTPCgi

Normally both classes can be combined into one for the available export. In VO this Design has been chosen because the class *HTTPExtensionContext* is also derived from *HTTPContext*. ISAPI and Extension-DLLs are not supported in Linux, therefore the design of the base class *HTTPCont* and the derived class *HTTPCgi* are not absolutely necessary. For experimenting with the OOP abilities of FlagShip, the retention of the design however was of great advantage. Here is a list of the prototypes of the imported classes:

```
PROTO CLASS HTTPCont
    PROTECT cMethod          AS STRING
    PROTECT cQueryString    AS STRING
    PROTECT cPath            AS STRING
    PROTECT aArgs            AS ARRAY
    PROTECT aEnvValues      AS ARRAY
    PROTECT aPostValues     AS ARRAY
    PROTECT nError           AS INTV
    PROTECT cBinRoot         AS STRING

 PROTO METHOD GetBinRoot  CLASS HTTPCont
 PROTO METHOD HTTPServerVariables ;
                             CLASS HTTPCont
 PROTO METHOD Init         CLASS HTTPCont
 PROTO ACCESS              CLASS HTTPCont
 PROTO ACCESS ServerPath  CLASS HTTPCont
 PROTO ACCESS QueryString CLASS HTTPCont
 PROTO ACCESS Error        CLASS HTTPCont
 PROTO METHOD GetParams   CLASS HTTPCont
 PROTO METHOD GetParamValue;
                             CLASS HTTPCont
 PROTO METHOD SetParamValue;
                             CLASS HTTPCont
 PROTO METHOD DelParamValue;
                             CLASS HTTPCont
 PROTO METHOD HTTPErrorMessage;
                             CLASS HTTPCont
 PROTO METHOD HTTPRequestItems;
                             CLASS HTTPCont
 PROTO METHOD HTTPMsg     CLASS HTTPCont
 PROTO METHOD GetServerVariable;
                             CLASS HTTPCont

PROTO CLASS HTTPCgi INHERIT HTTPCont
 PROTO METHOD  Init         CLASS HTTPCgi
 PROTO METHOD  Close        CLASS HTTPCgi
 PROTO METHOD  WriteImage  CLASS HTTPCgi
 PROTO METHOD  WriteContentFile;
                             CLASS HTTPCgi
 PROTO METHOD  Write        CLASS HTTPCgi
 PROTO METHOD  WriteClient CLASS HTTPCgi
```

The Test Program

So far one has always been talking about the “nearly complete” assumption of VO source code. Here is a trial balance:

- Due to the common Clipper roots of VO and FlagShip, we can fall back to a fundamental thing in common, their syntax.
- Missing functionality was delivered subsequently (*missing.fh, missing.prg, cmissing.c).*
- Differences that are present due to the different platforms became totally enclosed in the classes.
- Each platform has an application Framework that is somewhat different. This performs important initializations. In addition, the respective main module which publishes the results to the server is implemented here.
- For VO and FlagShip applications, the fact is common that the specific code for a CGI application is implemented in the method *HTTPResponse()* of the class *StdCgi.*

Same code for two platforms

Now the time has come for complete code-sharing. The VO example can be brought over 1:1, be compiled with FlagShip, and then be linked with the remaining modules to a Linux application:

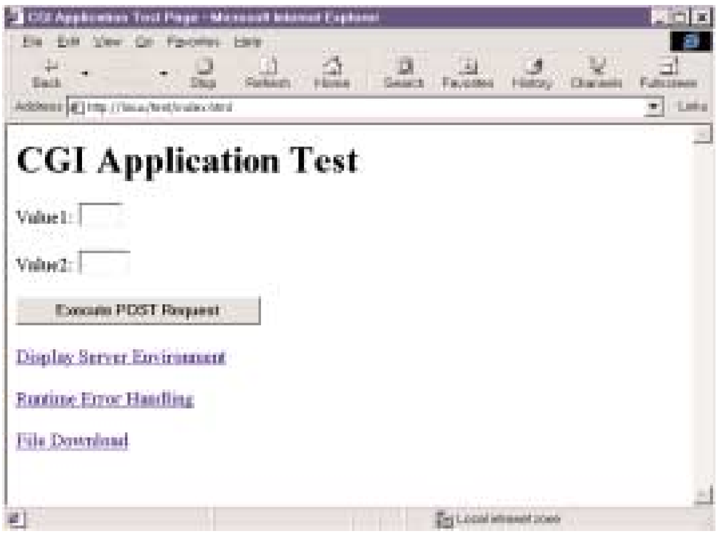
```
#include “missing.fh”
#include “httpcont.fh”
#include “httpcgi.fh”
#include “logfile.fh”

METHOD  HTTPResponse () CLASS StdCgi
    LOCAL cRet       AS STRING
    LOCAL cAction   AS STRING
    LOCAL x          AS USUAL

    cAction := lower(SELF:QueryString)

    DO CASE
    CASE cAction = “servervars”
        cRet := SELF:HTTPServerVariables()

    CASE cAction = “error”
        // a runtime Error is forced with
        // by an illegal operation
        x := 0
        cRet += x
    CASE ATC(“.zip”, cAction) > 0
```

```
        IF File(cAction)
            SELF:WriteContentFile(cAction,;
                  “application”, “zip”)
            cRet := “”
        ELSE
            cRet := SELF:HTTPMsg(“File “+;
                  cAction + “ not found”,;
                 “test.exe”)
        ENDIF
    OTHERWISE
        cRet := SELF:HTTPRequestItems()
    ENDCASE

    RETURN cRet
```

Naturally it concerns a simple example here. But also larger applidations can be converted problem-free with the described measures. A condition for it of course is the retention of the suggested program structure. The Html file *index.html* used for the VO example *votest.exe* serves for testing this:

```
<html>
<head>
<title>CGI Application Test Page</title>
</head>

<body>
<h1>CGI Application Test</h1>
<form action=”/cgi-bin/test/test.exe”
    method=”post” name=”TestForm”>
  <p>Value1: <input type=”text”
    name=”Value1” size=”4”></p>
  <p>Value2: <input type=”text”
    name=”Value2”  size=”5”></p>
  <p><input type=”submit”
    value=”Execute POST Request”></p>
</form>

<p><a href=”/cgi-bin/test/
  test.exe?SERVERVARS”>
  Display Server Environment
</a></p>

<p><a href=”/cgi-bin/test/
  test.exe?error”>
  Runtime Error Handling
</a></p>

<p><a href=”/cgi-bin/test/
  test.exe?test.zip”>
  File Download
</a></p>
</body>
</html>
```

Before the actual test, the file index.html must be copied into the weblist of the Linux server / <html Root>/test and the application test.exe must be copied into the listing / cgi bin/test /. Now it can be called via the Internet Browser by http://<Linux Server>/cgi bin/test/index.html . On condition of correct configuring of the web server Daemon, on the Linux side you should receive the following answer to this Request (fig. 2):



CGI Application Test

Value1:

Value2:

Execute POST Request

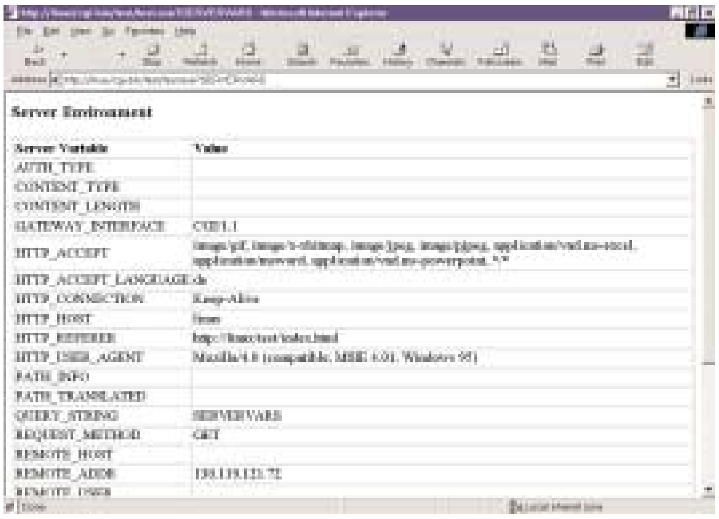
Display Server Environment

Runtime Error Handling

File Download

Fig.2: Data index.html on the Linux-Server

A click on the link “display server Environment“ grants us, as expected, information about the installed web software and its

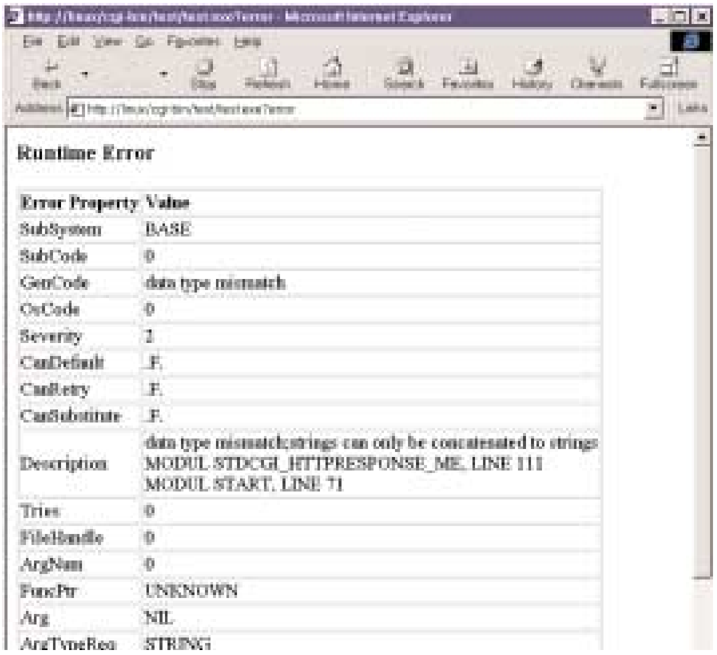


Server Environment

Server Variable	Value
AUTH_TYPE	
CONTENT_TYPE	
CONTENT_LENGTH	
GATEWAY_INTERFACE	CGI/1.1
HTTP_ACCEPT	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword, application/vnd.ms-powerpoint, */*
HTTP_ACCEPT_LANGUAGE	de
HTTP_CONNECTION	Keep-Alive
HTTP_HOST	linux
HTTP_REFERER	http://linux/test/index.html
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)
PATH_INFO	
PATH_TRANSLATED	
QUERY_STRING	SERVERVARS
REQUEST_METHOD	GET
REMOTE_HOST	
REMOTE_ADDR	130.119.123.72
REMOTE_USER	

Fig3: Server-Environment on the Linux-Side

Also the run time errors can be documented and prepared just as well under NT (Fig. 4):



Runtime Error

Error Property	Value
SubSystem	BASE
SubCode	0
GenCode	data type mismatch
OsCode	0
Severity	2
CanDefault	.F.
CanRetry	.F.
CanSubstitute	.F.
Description	data type mismatch;strings can only be concatenated to strings MODUL STDCGI_HTTPRESPONSE_ME, LINE 111 MODUL START, LINE 71
Tries	0
FileHandle	0
ArgNum	0
FuncPtr	UNKNOWN
Arg	NIL
ArgTypeReq	STRING

Fig4: Runtime errors edited in Html-Form

The remaining calls of the CGI-script *test.exe* (form processing, file download) work as expected as accurately on an NT server. The application was successfully ported with it.

The CGI-applidation is, in the case here, called test.exe. The file extension .exe is not compellingly necessary in UNIX to mark an executable file. This happens via appropriate characteristics. I personally think it is, however, quite practical to maintain such characteristics typical for DOS/Windows. Thus the file is already marked optically as an “executable file”.

In order to mark the script also for UNIX as executable, the instruction install can be used. Thus we have also arrived at the project file already mentioned. Here all necessary steps are agreed upon for the generation of the application including installation.

The project file

The available test routine consists of the following modules:

test.prg
missing.prg
cmissing.c
httpfunc.prg
httpcont.prg
httpcgi.prg
logfile.prg

You will find a possible variant for the appropriate project

file on the accompanying CD as file *Makefile*:

```
INSTALLDIR=/home/uwe/httpd/cgi-bin/test

CFLAGS=-I. -I../include
FSOPTS=-na -MStart -w1 -w2 -w3 -w4
LINKOPTS=-L. -L/usr/lib

all: test

test: test.o \
  ../lib/missing.o \
  ../lib/cmissing.o \
  ../lib/httpfunc.o \
  ../lib/httpcont.o \
  ../lib/httpcgi.o \
  ../lib/logfile.o
  FlagShip $(LINKOPTS) $(FSLIBS) \
            $(FSOPTS) -otest.exe $^

%.o: %.prg
  FlagShip $(CFLAGS) $(FSOPTS) \
            -rc -c -o$@ -q $<

install: test
  install -s -o uwe -g root -m 755 \
            test.exe $(INSTALLDIR)
```

Before the use of these scripts you must adjust only the installation listing (INSTALLDIR) and the call of the instruction install to your concrete conditions.

Result: CGI, what else!

The demand for a version of VO for Linux has lately been increasingly expressed. There are primarily CGI applications which must be ported from NT to Linux. My standard response in such cases was so far a reference to the FlagShip compiler. I decided to do a closer analysis of the product mentioned in order to be able to continue answering this with good conscience.

Beginning with the porting of the presented classes I converted several CGI applications. The present contribution is therefore rather an empiric report and does not claim any completeness. The compiler and other FlagShip utilities are only mentioned marginally. The contribution is written rather from the view of a specialist from beyond. Nevertheless I think that I can state the following with good conscience:

- FlagShip offers in principle everything that VO users need for the porting of their applications.
- In the case of use of the classes and functions contained in the project this porting is extremely facilitated.
- CGI applications can be ported very easily into UNIX. ASP, ISAPI, and other beginnings are not excluded from it. Whoever gets involved in these technologies should know that he commits himself exclusively to the NT environment.

The FlagShip compiler is a very professional, high performing and open development system. It is, besides Linux, available on all UNIX systems on the market. The technical support of the firm Multisoft is exemplary, inquiries and e-mails are answered quickly and precisely.

CA-Visual Objects and FlagShip form an interesting bundle for implementing platform Internet applications. In this sense the demand for a UNIX version of VO is unnecessary because there are already existing capabilities.

Literature

Stephen Genusa: Using ISAPI, Que, 1997 (ISBN 0-7897-0913-9)

M. Wielsch,J.Prahm,H. g. Esser: Linux internal, DATA Becker, 1999 (ISBN 3-8158-1292-5)

Dr. Olaf Borkner-Delcarlo: The Samba book, publishing house SuSE PRESS (ISBN 3-930419-93-9)

Multisoft data processing technology GmbH: FlagShip documentation, manual release: 4,4, 1999

Uwe Holz
Email: Uwe.Holz@ca.com

Georg S. Lorrig

Xbase++ SL 2 – the first impression

Shortly before this issue was sent to the press, Service Level 2 for Xbase++ was released. It brings in new functionality as well as substantial performance gains.

Speed

From the beginning, the modest performance of Xbase++ has been criticized by many users. Alaska has been able to achieve an improvement in this area with SL2. During the public beta phase there were messages from users enthusiastic about the speed increase. Here are some comparative measurements that I performed under Windows NT 4 SP 6 (Pentium III 560 MHz with 384 MB RAM). After creating the test data files the system was shut down and started again. Therefore side effects, such as caching, should be alike for all three programs. The standard DBF/NTX data base driver was used.

Functions	CA-Clipper 5.2e	Xbase++ SL1(178)	Xbase++ SL2 (xxx)
Indexing	00:52	02:02	00:47
Packing (index activated)	01:00	10:09	00:52
Packing (without index)	00:19	09:20	00:18

PACK was performed on the database where the first five records were deleted. If you now compare the results of SL1 and SL2 you are in for a big surprise at the SL2 speed boost.

However, not only the execution speed, but also the development cycle is turbo-charged: the Compile instruction *xpp* now accepts several program names. Since most programmers probably work with *PBUILD*, the benefit is the ability to compile several programs in one step.

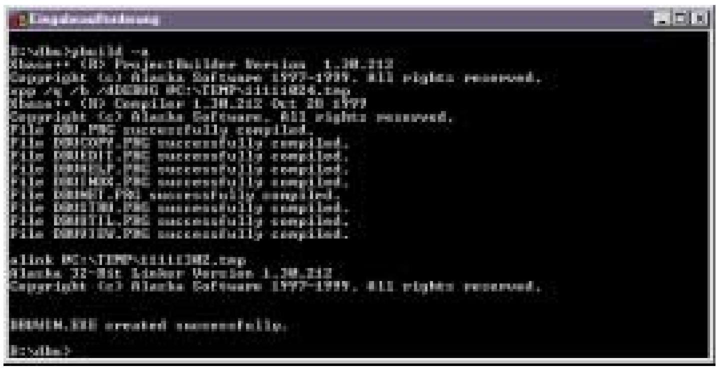
Installation

My first attempt to install failed but this was my own fault: I wanted to apply the international SL2 to a German installation. This is not possible for obvious reasons. A glance at the file *PDR.TXT* in the *\XPPW32* directory will tell you which language version is installed.

- It is important to regenerate all objects after the installation is complete. This also applies to all of your own libraries, etc.

PBUILD –a

takes care of that:



```
E:\dbm>pbuild -a
Xbase++ (R) ProjectBuilder Version 1.30.212
Copyright (c) Alaska Software 1997-1999. All rights reserved.
xpp /q /b /DEBUG @C:\TEMP\11111024.tmp
Xbase++ (R) Compiler 1.30.212 Oct 20 1999
Copyright (c) Alaska Software. All rights reserved.
File DBU.PRG successfully compiled.
File DBUCOPY.PRG successfully compiled.
File DBUEDIT.PRG successfully compiled.
File DBUHELP.PRG successfully compiled.
File DBUINDX.PRG successfully compiled.
File DBUNET.PRG successfully compiled.
File DBUSTRU.PRG successfully compiled.
File DBUUTIL.PRG successfully compiled.
File DBUVIEW.PRG successfully compiled.

alink @C:\TEMP\11111382.tmp
Alaska 32-Bit Linker Version 1.30.212
Copyright (c) Alaska Software 1997-1999. All rights reserved.

DBUWIN.EXE created successfully.

E:\dbm>
```

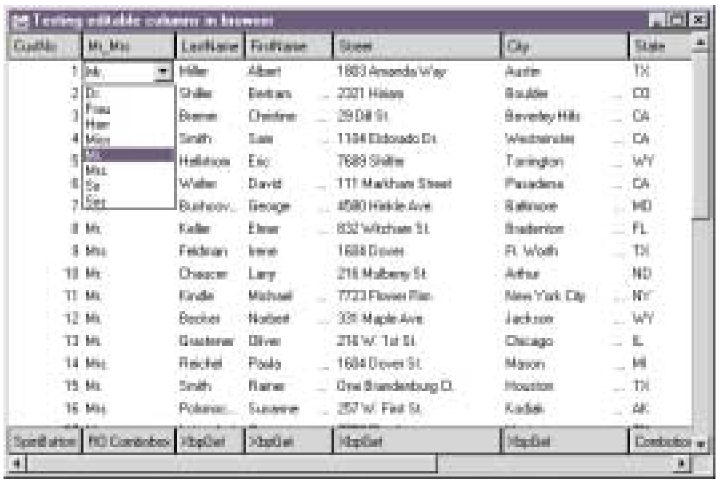
fig. *1*

Internals

- The database engine was revised for optimum speed and memory consumption. The effects already made clear in the remarks above.
- The popularity of the xBase languages probably has its origin in their flexibility. A pillar of this flexibility has always been the macro operator "&". With Clipper 5.0 the code blocks were introduced. In my own programs there is hardly a source file not using this feature. Normally code blocks represent pseudo-code, which still has to be interpreted. Now in certain situations the Xbase++ compiler is able to produce native code, which the code block refers to. This should likewise lead to better performance.
- Apart from *DBFDBE* and *NTXDBE*, now ***DELDBE*** and ***SDFDBE*** are also loaded according to the standard. Whoever uses an adapted version of the *DBESYS.PRG* should understand this issue and make appropriate modifications.

Extensions

- Xbase parts can now be configured with foreground and background colors. This creates for example the possibility to mark field attributes, such as 'field input necessary', by the choice of color.
- The GUI GET element *XbpGet* has now a pre- and a post-validation added. In order to handle the focus change, a new *XbpGetController* class was introduced. This class comes with source code, thus you get a good opportunity to study some Xbase++ internals.
- The way from the familiar but lusterless text mode to the 'brave new GUI world' was sometimes very thorny. While in the text mode you do not need to write a lot of code in order to be able to edit an active cell in a TBrowse object, this was inconceivable in the GUI environment. *XbpBrowse* is a class derived from *XbpStatic*, which is 'static', as the name suggests. With SL2 this also works in *XbpBrowse*. But the guys (& gals?) from Alaska didn't stop here: it is possible now to assign specific GUI controls for each column individually. I.e., you can use a XbpSpinButton to select an amount, or the country code can be selected from a XbpListBox. The possibilities opening here are really fascinating:



Testing editable columns in browser

CustNo	Mr_Mrs	LastName	FirstName	Street	City	State
1	Mr.	Miller	Albert	1883 Amanda Way	Austin	TX
2		Shiller	Bertram	2321 Hiawa	Boulder	CO
3		Bremer	Christine	29 Dill St.	Beverley Hills	CA
4		Smith	Sam	1194 Eldorado Dr.	Westminster	CA
5		Hellstrom	Eric	7689 Shiller	Torrington	WY
6		Walter	David	111 Markham Street	Pasadena	CA
7		Bushoov...	George	4580 Hinkle Ave.	Baltimore	MD
8	Mr.	Keller	Elmer	832 Witcham St.	Bradenton	FL
9	Mrs.	Feldman	Irene	1688 Dover	Ft. Worth	TX
10	Mr.	Chaucer	Larry	216 Mulberry St.	Arthur	ND
11	Mr.	Kindle	Michael	7723 Flower Rec.	New York City	NY
12	Mr.	Becker	Norbert	331 Maple Ave.	Jackson	WY
13	Mr.	Gastener	Oliver	216 W. 1st St.	Chicago	IL
14	Mrs.	Reichel	Paula	1684 Dover St.	Mason	MI
15	Mr.	Smith	Rainer	One Brandenburg Ct.	Houston	TX
16	Mrs.	Pokorsc...	Suzanne	257 W. First St.	Kodiak	AK
SpinButton	RO Combobox	XbpGet	XbpGet	XbpGet	XbpGet	Combobox

Dr.
Frau
Herr
Miss
Mr.
Mrs.
Sr
Sra

Pic.2, CellEdit

- Most Xbase++ parts are 'genuine' GUI controls, like the ones in many other Windows programs. With the *XbpColumn/XbpBrowse* combination Alaska created a new version of these controls. *XbpColumn* was extended and now offers the possibility of 'building' your own browser objects. 'Browse' does not necessarily have to be a tabular view of a database or an array.
- The *DacPagedDataStore* class is totally new. It is the basic structure behind the new *XbpQuickBrowse* and it permits cached access to tabular data, no matter whether the source is a database or a two-dimensional array.
- As the name suggests, allows *XbpQuickBrowse* to implement graphic Browser with minimum effort. This one is not as configurable as *XbpBrowse*, but it fulfills its purpose in quick and dirty solutions. It is also well suitable for prototyping.

Prospects

The speed increase is naturally the first thing to mention. Next come extended possibilities, like those offered by the *XbpBrowse* and *XbpColumn* classes.

The SL2 version points out very nicely that there is a large potential for Xbase++ and that Alaska is willing to use and extend this potential. 'Under the hood' of SL2 the first glimpses on Xbase++ 2.0, which will lean even more towards Client/Server, are already visible. Brace yourself for many pleasant surprises to follow.

Georg S. Lorrig
Email: georg@lorrig.com

Web Applications with Alaska Web Application Adapter (WAA). Part II.

Basics for the web application examples

The WAA of Alaska provides classes to create dynamic HTML pages with forms and edit controls. However, current standards demand more complex design and dynamic generation of web pages would require intimate knowledge of HTML. Besides, the structure of the pages would not be that efficient. That is why I use NetObjects Fusion to create the entire site. This tool lets you simply paint the whole site and learn the necessary HTML and Java skills along the way. The following examples show how to transform the 'painted' HTML pages into dynamic pages to be used with WAA. The conference CD contains the trial version of NetObjects Fusion 4.0. The directory UserSite\waa99_NOx contains NetObjects project files for the Home_Nox directory.

At the moment, web applications run only with XBase++ version 1.20.178. The directories Home_NO1 - Home_NO5 and Source_NO1 –Source_NO5 correspond to individual examples. For each example the respective **Home_Nox** directory should be copied into **Home** and the directory **Source_NOx** into **SOURCE** and **buildd.bat** should be called in SOURCE. The included XWEB.DLL is valid up to 01.03.2000.

You should make necessary changes in the file Waa99.ini in the WaaServer directory. The directory Tools contains the trial version of NetObjects fusion 4.0 and the WaaServer installation for the Apache Web Server.

To run applications Apache or another web server should be installed. The Home directory must be created in WaaSession\Home. The directory Apache_Config should contain appropriate Apache configuration files with name entries to be adjusted. The WAA Server and WAA Gateway are already in appropriate directories. First the web server must be started. In order to start the WAA Server the file waa.cmd or waadbg.cmd (for debug mode) must be called. The entry WAA_HOST in this file should be adapted first.

Installation and necessary adjustments

Installation of examples does not create a Web Server! The installation of the WAA Server in the WaaSession\Tools\WAA directory includes the installation of the Apache Web Server. The WAA patches in the same directory must be installed as well!

Please do not install the WAA Server into the directory WaaSession. This directory contains WAA Server with session-specific files already installed!

At the end of the installation the question "enable HTML help?" should be answered affirmative for the XWeb help to be available. XWeb help requires MS Internet Explorer installed.
After the installation the following adjustments should be made:

- In the directory WaaSession: in WAA.BAT and WAADBG.BAT the entry "SET WAA_HOST=" enter your host name here.
- In the directory WaaSesion\WaaServer: in WAA99.ini the entries DBF=, Home= and Script= must be adapted.
- In the directory WaaSesion\Home\cgi-bin: in WAA1GATE.CFG the entry "host name =" must be completed with your host name.
- It is possible that some images will be not visible when NetObjects projects in the directory UserSite are opened. In that case select the appropriate image in the Image File Properties/Browse dialog.

Web applications with visually designed HTML files

Entry form (example 1)

This example contains a log-in page and a response page. The Site was entirely painted with NetObjects.

Some special features for the entries in NetObjects for communication with the WAA:

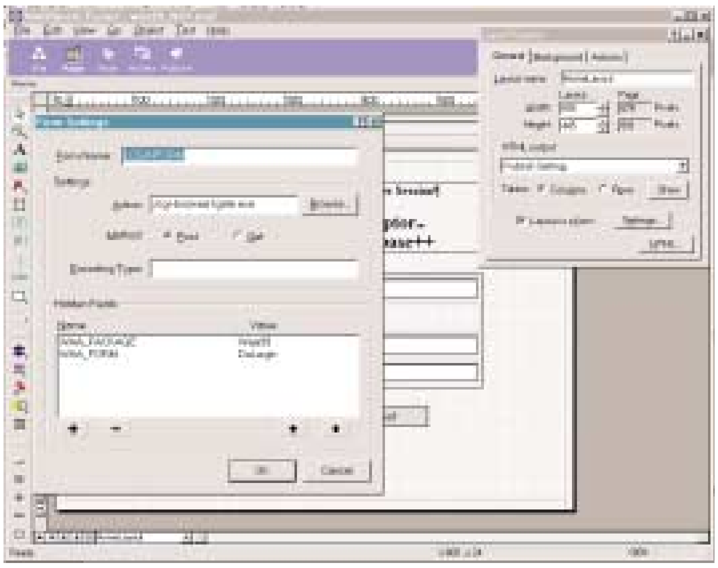


Figure 1NetObjects fusion adjustments

- In the layout properties „layout as a form" must be checked
- In form settings the access to WAA gateway must be entered under „action": /cgi-bin/waa1gate.exe.
- Hidden variables WAA_PACKAGE and WAA_FORM must be entered.
- In properties for input fields the name of the field that later will be used with oHTML:getVar(...) must be entered. The names are *case sensitive!*

The O.K. button is the pages submit button. Clicking this button in the browser initiates the execution of what has been entered under „Action". In our case the WAA Gateway is called, which passes on the variable values of the page to the WAA Server. The WAA Server will determine based on the WAA_FORM value which DLL function to call.

Path adjustments for the web application

In order to place correct paths into the program code they are stored in an INI file. The INI file must be located in the WAA Server directory. The section and entry names of the INI file are fixed:

```
[SiteData]
; absoluter Pfad zu DBFDBF=\devcon99\data_dbf

;absoluter Pfad zu den HTML-Dateien entspr. ;dem Home-
Verzeichnis
Home=\devcon99\home

;absoluter Pfad zu den (Java)-Script-;Dateien
Script=\devcon99\data_script

;relativer Pfad vom WaaServer zu den Image-;Dateien
SrvImage=..\home\image

;relativer Pfad vom WaaGateway zu den ;Image-Dateien
GwImage=../image

;relativer Pfad vom Home-Verzeichnis zum ;cgi-bin
Verzeichnis
CGI=/cgi-bin
```

The paths are internally used to open databases and read HTML files. The SrvImage path entry can be used to check for the presence of image files. The GwImage path entry is used to dynamically bind image files into HTML code.

Creating a dynamic HTML page

In our example the DLL function DoLogin is called.

```
FUNC DoLogin(oHtml, oContext)
LOCAL cVName  := oHTML:getVar(“VNAME”)
LOCAL cAnrede := oHTML:getVar(“ANREDE”)
LOCAL cNName  := oHTML:getVar(“NNAME”)
LOCAL cPass   := oHTML:getVar(“PWORD”)

oEnv := SetHomeEnv(WAA99, “waa99.ini”)
if ! oEnv:OpenDataSource(“CUSTOMER”,
   {“CUSTOMER”}, “PWORD”, “FOXCDX”, oHtml)
    SendMsg(oHTML, “Fehler”, “Kann
             Datenbank nicht öffnen”)
   RETURN True
endif

usw.....
oEnv:CloseDataSource()
cText := “Guten Tag”;
         +cAnrede+cVName+cNName+”!”


// lade HTML-Seite
cHtml := oEnv:LoadHtmlFile(“mainmenu.html”,
                             False, oHtml)

// ersetze Tag
cHtml := strtran(cHtml, “#USERINFO”, cText)

// sende dynamische page
oHTML:header(cHtml)
RETURN TRUE
```

The contents of a variable field are queried by an oHTML:getVar(“VNAME“) call. In this case these are the contents of a control on the form. SetHomeEnv() reads the INI file and returns the Environment object. Methods of the Environment object use path entries from the INI file. SetHomeEnv() and the Environment methods are contained in the XWeb.DLL. The method LoadHtmlFile() reads the painted HTML page into a string variable. On this page the text “#USERINFO“ was entered as the user tag. This user tag is now replaced by our response text. The method Header() of the HTML object receives the HTML string. If TRUE is returned the HTML string of the HTML object is sent as a page to the Browser.

Form validating (application example No2)

The example No2 looks pretty much like No1, but here we want to add input validation.

Validating on the server side

The simplest type of validating takes place on the server side in the SOURCE code:

```
LOCAL cVName  := oHTML:getVar(“VNAME”)
LOCAL cNName  := oHTML:getVar(“NNAME”)
LOCAL cPass   := oHTML:getVar(“PWORD”)
if empty(cPass)
    SendMsg(oHTML, “Eingabe-Fehler”, ;
        “Bitte geben Sie ein Passwort ein”)
  RETURN True

elseif empty(cNName) .or. empty(cVName)
oEnv:CloseDataSource()
  SendMsg(oHTML, “Eingabe-Fehler”, ;
  “Bitte geben Sie Vor- und Nachnamen ein”)
  RETURN True
endif
```

The content of the variable is determined by calling oHTML.getVar(). If it is empty a message is sent to the Browser. SendMsg() is contained in the XWeb.DLL.

The disadvantage of this method is the fact that the input form must always be sent first. For validating that requires database access this is the only way. For trivial validating, however, there is a possibility of validating directly on the client side.

Validating on the client side with Java Script

For validating on the client side a Script language is necessary. In the examples we use exclusively Java Script.

In the preceding section we learned that the “OK“ button is a submit button which will send the request to the WAA Gateway. In Java Script there are so-called “Events“ e.g. onClick. The onClick is triggered when certain HTML controls are clikked or operated. In HTML syntax it looks this way: < onClick=return “LoginCheck()>“, which means that the Script function LoginCheck() is called when the button is operated. If this function returns FALSE, no submit is executed.

HTML tag for the submit button with Java Script:

```
<INPUT TYPE=SUBMIT NAME=”pbOk” VALUE=”Ok”
     ID=”FormsButton1”
     onClick=”return LoginCheck()”>
```

```
<script language="JavaScript">  <!—
function LoginCheck() {
var checkStr =
   document.LOGINFORM.PWORD.value ;

if ( checkStr.length < 4 ) {
  alert("Bitte geben Sie das Passwort mit
          mindestens 4 Zeichen ein") ;

  document.LOGINFORM.PWORD.focus() ;
  return (false) ;
}
return (true) ;
} //—> </script>
```

Java-Script in a HTML file:

```
<......onClick=this.form.WAA_ACTION.value="
DoOrderForm">
```

The onClick instruction is simply entered in NetObjects into the Properties of the Button tag and Java Script can be inserted e.g. as file into the Properties of the layout form. Placing within the HTML code is executed automatically.

Internet Shopping (example 3)

With this application a fundamental change is introduced. Each page makes calls only to the DLL function "DoDispatch()". There the necessary function is selected from the CASE statement branch. For this purpose a hidden variable WAA_ACTION is created for each page. This variable receives at run-time the name of the function to be executed in DoDispatch. The advantage of this approach is that you can use in a form as many DLL functions as you like. This is implemented with Java Script too and uses the already introduced onClick event.
The function DoArticle() produces a dynamic page to display all articles and their availability. For this purpose two pages are created in NetObjects. The Article is the HTML page dynamically filled with data. The Article is designed only to generate the HTML to display the articles. The hidden variable WAA_CARGO that already stores the user ID passed by DoLogin(), is sent to every page. This way there is no need to create a Cookie for ordering processes.

```
FUNC DoArticle(oHtml, oContext)
LOCAL cHtml, nPos, oEnv
LOCAL cCargo := Html:getVar("WAA_CARGO")

oEnv := SetHomeEnv(WAA99, "waa99.ini")

if ! oEnv:OpenDataSource("COMPUTER",
       {"COMPUTER"}, "PRODUCER",
        "FOXCDX", oHtml)
   SendMsg(oHTML, "Daten-Fehler", ;
           "Kann Datenbank nicht öffnen")
   RETURN True
endif

cHtml := oEnv:LoadHtmlFile("artikel.html",;
                           False, oHtml)

// Beginn für Dateneintrag
nPos  := GetDataPos(@cHtml)
dbGotop()
do while ! EOF()
  // Daten hinzufügen
  nPos := AddData(oEnv, nPos, @cHtml, ;
          AsString(recno()), ;
         field->partno, ;
          field->producer, ;
          field->computer, ;
          field->cputype, ;
          field->cpucount, ;
          field->price, ;
          field->image)
  dbSkip()
enddo
oEnv:CloseDataSource()

// Cargo für User-Werte setzen
SetValue(@cHtml, "WAA_CARGO", cCargo)

// sende dynamische page
oHTML:header(cHtml)
RETURN True
```

GetDataPos() determines the position of the user tag #DATA that marks the position from which data is inserted. AddData() inserts data into the HTML string.

```
// Suche nach dem Tag </TABLE> nach dem
// Usertag #Data um Daten einzufügen
// <TD WIDTH=149><P>#DATA</TD></TR></TABLE>

STATIC FUNC GetDataPos(pcHtml)
LOCAL nPos
nPos   := at("#DATA", pcHtml)

// #Data Tag entfernen
pcHtml := strtran(pcHtml, "#DATA", &nbsp;")


// suche nach Tabellen-Ende
nPos   := strAtX("</TABLE", pcHtml, nPos)

// Position um Daten einzufügen
nPos   -= 1

RETURN nPos
```

The HTML code was generated the page designer and is inserted into the HTML string line by line.

```
// Tabellenzeile ab nPos hinzufügen und mit // Daten
füllen
STATIC FUNC AddData(oEnv, nPos, pcHtml,
       cRec, cPartNo, cProducer, cComputer,
       cCpuType, nCpuCount, nPrice, cImage)
LOCAL cTemp:="", cChkName

// checkbox  Name pro Satz: Check1...Checkn
cChkName := "Check"+cRec
cTemp += '<TR>'+_LF
cTemp += '<TD WIDTH=67><P><DIV
                           ALIGN="CENTER">'+_LF
cTemp += '<TABLE WIDTH=13 BORDER=0
         CELLSPACING=0 CELLPADDING=0>'+_LF
cTemp += '<TR>'+_LF
cTemp += '<TD><INPUT ID="FormsCheckbox1"
            TYPE=CHECKBOX NAME="'+cChkName;
         +'"VALUE="'+cRec+'"></TD>'+_LF
cTemp += '</TR>'+_LF
cTemp += '</TABLE>'+_LF
cTemp += '</DIV></TD>'+_LF

// Daten
cTemp += '<TD VALIGN=TOP
```

```
          WIDTH=71><P>'+cPartNo+'</TD>'+_LF
cTemp += '<TD WIDTH=229><P>' ;
           +cProducer+'<BR>'+cComputer ;
           +'<BR>'+cCpuType ;
           +'<BR>'+AsString(nCpuCount);
           +'</TD>'+_LF
cTemp += '<TD VALIGN=TOP WIDTH=82><P>' ;
         +AsString(nPrice)+'</TD>'+_LF

if file(oEnv:ServerImagePath()+cImage)
  cTemp += '<TD VALIGN=TOP WIDTH=149><P>' ;
     +'<IMG ID="Picture1" HEIGHT=100
                         WIDTH=100 SRC="' ;
     +oEnv:GatewayImagePath()+cImage ;
     +'" VSPACE=0 HSPACE=0 ALIGN="TOP"
                  BORDER=0>&nbsp;</TD>'+_LF

else
  cTemp += '<TD WIDTH=149>&nbsp;</TD>'+_LF
endif
cTemp  += '</TR>'+_LF
pcHtml := stuff(pcHtml, nPos, 0, cTemp)
nPos  += len(cTemp)
RETURN nPos
```

The name and value of the check box includes the current record number. This way it is possible to determine for a marked check box the respective record number.

The Order button click in the Article Browser will result in the "DoOrderForm" function call over DoDispatch().

```
// Bestellformular ausgeben
FUNC DoOrderForm(oHtml, oContext)
LOCAL cHtml, nPos, oEnv
LOCAL cCargo := oHtml:getVar("WAA_CARGO")
LOCAL cOrder:="", cChkName
LOCAL cCustID

oEnv := SetHomeEnv(WAA99, "waa99.ini")
if ! oEnv:OpenDataSource("COMPUTER",
       {"COMPUTER"}, "PRODUCER", "FOXCDX",
          oHtml)
    SendMsg(oHTML, "Daten-Fehler", ;
           +"Kann Datenbank nicht öffnen")
   RETURN True
endif
cCustID := GetCargo(oHtml)[1]
dbGotop()

// alle markierten Einträge suchen
do while ! EOF()
  cChkName := "Check"+AsString(recno())
  if ! empty(oHtml:getVar(cChkName))
    cOrder += field->partno+" " ;
           +rtrim(field->computer)+_LF
  endif
  dbSkip()
enddo
cHtml := ;
     oEnv:LoadHtmlFile("bestellung.html", ;
     False, oHtml)
SetListSelect(@cHtml, "ANREDE", ;
              field->anrede)
SetValue(@cHtml, "VNAME", field->vname)
SetValue(@cHtml, "NNAME", field->nname)
SetMleValue(@cHtml, "mleORDER", cOrder)
SetValue(@cHtml, "WAA_CARGO", cCargo)

oEnv:CloseDataSource()
oHTML:header(cHtml)
RETURN True
```

Within the DO WHILE loop the state of each checkbox on the page is verified. Since the checkbox name contains the record number, you can use it as a reference. Analyzing the check boxes on the server side has a big disadvantage that all records must be skipped and all check box states determined. This results in a slow response. This behavior is avoided in the next application by performing the check box validation on the client side.

Next, generates the function the dynamic Order page. The page contains controls with the names VNAME, NNAME etc… Their values are inserted into the HTML string by the SetValue() function.

Internet Shopping (example 4)

As already mentioned in chapter 3.3, the state of the check boxes is determined in this example on the Client side. Thus the response is substantially improved. In order to achieve this, Java Script must be inserted into the HTML string by the function AddData().

```
// Tabellenzeile ab nPos hinzufügen und mit
// Daten füllen
STATIC FUNC AddData(oEnv, nPos, pcHtml,
       cRec, cPartNo, cProducer, cComputer,
      cCpuType, nCpuCount, nPrice, cImage)
LOCAL cTemp:="", cChkName, cScript

// checkbox  Name pro Satz: Check1...Checkn
/* <td><input type="Checkbox"
              name="Check34" value="34"
          onclick="CheckSelected(this.form,
          this.form.Check1)"></td>
*/
cChkName := "Check"+cRec
cScript  :=
         'onclick="CheckSelected(this.form,
         this.form.'+cChkName+')"'
cTemp += '<TR>'+_LF
cTemp += '<TD WIDTH=67><P>;
          <DIV ALIGN="CENTER">'+_LF
cTemp += '<TABLE WIDTH=13 BORDER=0
        CELLSPACING=0 CELLPADDING=0>'+_LF
cTemp += '<TR>'+_LF

cTemp += '<TD><INPUT ID="FormsCheckbox1"
         TYPE=CHECKBOX NAME="'+cChkName+'"
         VALUE="'+cRec+'"  ';
         +cScript+'></TD>'+_LF
cTemp += '</TR>'+_LF
cTemp += '</TABLE>'+_LF
cTemp += '</DIV></TD>'+_LF
```

Each check box has a corresponding onClick Java Event. Clicking the check box will trigger a call to the Java function CheckSelected.

The Script for the CheckSelected() function is inserted into the HTML string by DoArtikel():

```
dbGotop()
do while ! EOF()
  // Daten hinzufügen
  nPos := AddData(oEnv, nPos, @cHtml,
      AsString(recno()), ;
      field->partno, field->producer, ;
      field->computer, ;
      field->cputype, field->cpucount,;
      field->price,   field->image)
  iCnt++
  dbSkip()
enddo
oEnv:InsertScript(@cHtml, ;
        "CheckSelected.js", ;
   oHtml, {{"#ELEMENTS", AsString(iCnt)}})
```

In addition a hidden variable WAA_SELECTED is created in the HTML code. This variable is used to store the select string which consists of the values (record numbers) of the marked check boxes and is determined in the Java function CheckSelected(). This string is then analyzed in DoOrderForm().

Special features in example No4

Setting the input focus

When an HTML form is displayed, the input focus should be on an input control. This can be achieved in Java Script in the HTML code. In NetObjects it is done in the layout Properties/HTML/Between head tags, where LOGINFORM is the name of the form tag and PWORD the name of the HTML form control to be focused.

```
<script language="JavaScript">  <!—
function SetOpenFocus(){
  document.LOGINFORM.PWORD.focus() ;
} //—> </script>
```

The function SetOpenFocus() must be called when the page is loaded. For this purpose the Java onLoad Event is inserted into the Body tag.

```
<BODY .... onLoad="SetOpenFocus()" ....>
```

Properties/HTML/Inside Body tag.

Call to Java Script on the server

Java Script can also be executed without inserting of the Script directly into the HTML code. Instead the HTML code should provide the name of the script file.

```
<script language="JavaScript"
        src="./java/daytime.js"
        type="text/javascript"> </script>
```

This Java Script displays the date and the daytime. In this example the function is called by clicking the clock image. To do so an image is created in NetOjects and the following entry in the Properties HTML/Inside tag is made:

```
onClick="SayDayTime()"
```

SayDayTime() is a Java Script function contained in the file./java/daytime.js. The script file should be placed within the Home directory of the web server.

Graphics as Submit Buttons

On the page MainMenu there are two graphic images which are used as Submit Buttons. This is accomplished by the following onClick entry in NetObjects graphic image Properties HTML/Inside:

```
onClick="SubmitForm('DoArticle')"
```

SubmitForm() is a Java Script function that takes the XBase DLL function name as parameter. The Java Script function is entered in the layout Properties HTML/Between Head tags:

```
function SubmitForm(cAction){
   document.MENUFORM.WAA_ACTION.value =
          cAction ;
   document.MENUFORM.submit() }
```

The hidden variable WAA_ACTION is used to store the XBase function name and the Submit() function will make a call to the WAA Gateway. This is the same procedure as using a Submit Button.

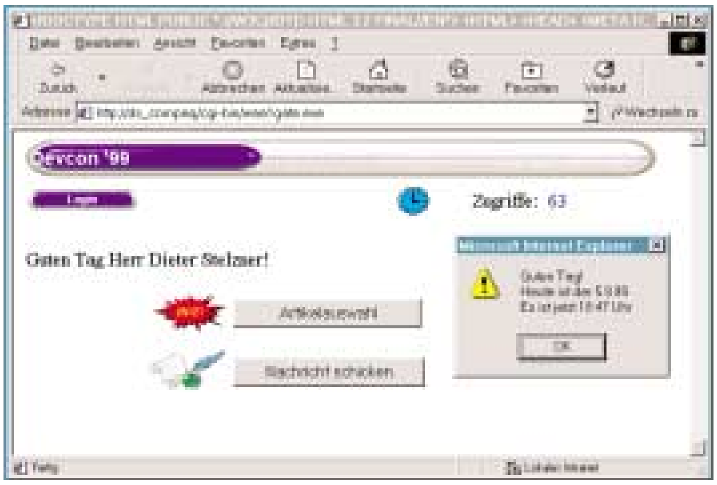


Figure 4: Graphics as Submit Controls

Internet Shopping with Frame page (example 5)

The problem with article selection in previous examples was that the Order button becomes visible only after scrolling down. In this example the article selection consists of 3 frames with only the body frame (article representation) to scroll. In the left and in the upper frames there are different type order buttons that are always visible.
The article page code is in four HTML files: artikel.html, left_artikel.html, header_artikel.html and body_artikel.html. The artikel.html contain the frame definition:

```
<HTML>
<HEAD>
<TITLE>Artikel</TITLE>
<FRAMESET COLS="152,*">
<FRAME NAME="left"
    SRC="./left_artikel.html"
    SCROLLING=AUTO MARGINWIDTH="2"
    MARGINHEIGHT="1" BORDER=5 NORESIZE>

<FRAMESET ROWS="121,*">
<FRAME NAME="header"
    SRC="./header_artikel.html"
    SCROLLING=AUTO MARGINWIDTH="2"
    MARGINHEIGHT="1" BORDER=5 NORESIZE>
<FRAME NAME="body"
    SRC="./body_artikel.html"
    SCROLLING=AUTO MARGINWIDTH=2
    MARGINHEIGHT=2>
```

```
</FRAMESET>
</FRAMESET>
</HEAD>
</HTML>
```

Each frame behaves like an independent HTML page. All frame pages have one form tag of the same name and same hidden variables. To make a Java Script function call within a frame, you must define it in the frame HTML file.

In the left Frame the Order button is a graphic image that has a Java Script function assigned as the onClick event. This in turn will call the Submit() function. On the XBase page the variables will be queried. However, the query with:getVar(...) always refers to the form sending the Submit, and in our case the variables would be empty. The problem is that the article selection is made in the Body Frame and the result string is only filled in by the hidden variables of the Body Frame. The same applies to the WAA_CARGO variable declared in the XBase code.

There are now 2 possibilities:

The frame control that should send the Submit would instead pass it on to the Body Frame.

The values of the hidden variables of the Body Frame are copied into the variables of the Frame sending the Submit.

For ordering we choose the first case. This looks like this: the Submit Graphic in the left Frame or the Submit Button in the Top Frame use the onClick Event:

```
<...onClick=”DoBestellen()”>
```

The Java Script function DoBestellen() (must exist in both Frame HTML files) calls an onClick Event for the Submit Button of the Body Frame:

Possibility 1.

```
function DoBestellen()  {
parent.body.document.LAYOUTFORM.pbBestellen
      .click()}
```

Possibility 2: copying the values of the variables

```
function DoBestellen()  {
var oLForm = parent.left.document.LAYOUTFORM ;
var oBForm = parent.body.document.LAYOUTFORM ;

oLForm.WAA_ACTION.value = “DoOrderForm” ;
oLForm.WAA_CARGO.value  =
      oBForm.WAA_CARGO.value;
oLForm.WAA_SELECTED.value  =
      oBForm.WAA_SELECTED.value ;
oLForm.submit() }
```

Now we are ready to produce Dynamic Frame HTML pages.

```
FUNC DoArticle(oHtml, oContext)
LOCAL cHtml, nPos, oEnv, iCnt:=0
LOCAL cFileB, cFileL
LOCAL cStamp
LOCAL cCargo := oHtml:getVar(“WAA_CARGO”)

oEnv := SetHomeEnv(WAA99, “waa99.ini”)
// temp. Html-Dateien löschen
oEnv:EraseTempHtml(“xx*.html”)

  usw....

cHtml := ;
    oEnv:LoadHtmlFile(“body_artikel.html”,False,oHtml)

  usw....

cStamp := “xx”+AsString(seconds())

// temp. Dateinamen für body_artikel.html =
//        xx123456body_artikel.html

cFileB := cStamp+”body_artikel.html”
oEnv:WriteHtmlFile(oEnv:HomePath()+cFileB, cHtml)

// temp. Dateinamen für left_artikel.html =
//        xx123456left_artikel.html
// Verweis auf body mit temp-namen ersetzen
cHtml  := ; oEnv:LoadHtmlFile(“left_artikel.html”,;
    True, oHtml)
cHtml  := strtran(cHtml,body_artikel.html”, ;
   cFileB, , 1)
// left dateinamen umbenennen
cFileL := cStamp+”left_artikel.html”
oEnv:WriteHtmlFile(oEnv:HomePath()+cFileL,;
                 rtrim(cHtml))
// in dynam. Page die Verweise auf die //temp. Dateien
ersetzen
cHtml := oEnv:LoadHtmlFile(“artikel.html”,, oHtml)
cHtml := strtran(cHtml, ;
        “body_artikel.html”, cFileB, , 1)
cHtml := strtran(cHtml, left_artikel.html”,cFileL,, 1)

// sende dynamische page
oHTML:header(cHtml)
RETURN True
```

As discussed earlier, in our case the individual Frame HTML pages are called by artikel.html. However, since the dynamic data is shown in the Body Frame, the Body Frame page must be written over as HTML file for each service request. Because several users can send requests at the same time, the article.html should be able to distinguish between several Body Frame pages. For that reason each temporary Frame page file gets a timestamp as a part of the file name. The file artikel.html is loaded as dynamic string in which the temporary Frame file names are inserted. To prevent the hard disk ‘pollution’ all temporary HTML files older than 1 minute are deleted at the function start: oEnv:EraseTempHtml(“xx*.html “).

Conclusion

Past experience has shown that design of complex Web sites requires substantially more time. Development cycle will increase even more if you want your Web application to run on any Web Browser. It is reasonable to design for MS Internet Explorer 4.0 and Netscape 4.0, which became de facto standards. Netscape is less compatible with HTML 4 compared to MS IE, which is not fully compatible too. Each browser has specific extensions, which in practice means that some nice things possible with one Browser may not work with another. The point is to test the application with different browsers and find browser-specific solutions, which of course can take a lot of time.

In the above examples we learned how easy it is to create web applications using XBase++ and WAA Server from Alaska Software and an HTML form designer. This also works without knowing much about HTML. The automatic HTML code generation makes it possible to grow into “HTML“ by analyzing it.

DS-Datasoft, Dieter Stelzner

Undocumented VO

By Rod da Silva

In my column last issue I began what I promised would be a series of articles uncovering the mysteries of VO's superior class/object model. While I have lots to talk about on that subject, I have decided to postpone that information until next time so that I can describe a new undocumented feature that has found its way into the recently released VO 2.5a patch. I consider the new feature important enough and useful enough to virtually every serious VO developer, that I thought it warranted preempting my scheduled column on VO's internal class structures. Indeed if you have been bitten at any time with a seemingly random VO dynamic memory error such as the dreaded 5333, you will want to pay special attention to this month's article.

As always, I hope you find the information that follows useful. However, I would be remiss if I did not restate my standard disclaimer that the information provided here is by definition undocumented (or poorly documented at best) and is not in any way supported by Computer Associates. If you use any, or all of this information, you do so at your own risk. Having said that, I personally feel confident that given the usefulness of what I am about to show you, I would be very surprised if CA development will ever remove this particular feature from the product.

HEEEELLLLLPPPPPP!!!!!!

Virtually everybody's seen them. They are almost impossible to reproduce. Horror stories about them in the various on-line VO forums abound. People plead for help. CA development insists the problem is in your code. It is of course, the 5333 error (previously know as the 4660). These errors—effectively system trapped GPFs—have caused no end of trouble for a number of otherwise excellent VO applications, especially since VO 2.5 was released. And no one seems to know where they come from or more importantly how to get rid of them.

The problem with these errors, as anyone who has tried to locate the cause of one will tell you, is that they are almost impossible to recreate at will. They appear on some machines (often the client's machine) almost immediately, albeit randomly, while never showing up on other machines (like the developer machines). Furthermore, the cause of them is almost never anywhere near where they actually surface (often 1000s of lines of code away), making finding them at lot like looking for a needle in a haystack.

Well, if you are being plagued by these errors, the recently released VO 2.5a patch offers new hope for locating and squashing them once and for all!

A Time Bomb Waiting To Go Off

In order to give context to this discussion we are going to require a dynamic memory bug. Consider the following code:

```
FUNCTION Start
  LOCAL o AS OBJECT
  LOCAL p AS PTR
  o := Error{}   // some Object
  p := PTR(_CAST, o )
// static memory holds pointer to dynamic type  ? o,
“,”, p   // addresses are the same

  RegisterKID(@p,1,.F.)  // register the KID
  CollectForced()
// moves object, pointer updates automatically
? o, “,”, p   // addresses are still the same!!
  WAIT
RETURN
```

This code seems to work great. It demonstrates the use of the RegisterKID() function. By registering the static pointer p, which points to the dynamic object o, with the VO runtime, when VO moves the object in memory the static pointer's value will automatically be updated to point to the new position of the dynamic object. This can be clearly seen in the following output:

```
{(0x0078)0x01040138} CLASS ERROR , 0x01040138
{(0x0078)0x02041124} CLASS ERROR , 0x02041124
Press any key to continue...
```

Notice how the address values of the object and the pointer match on the first line (i.e.; 0x01040138) and then again on the 2nd line (i.e.; 0x02041124) after the CollectForced() call forced the object to move in memory. This shows how the registered pointer was automatically updated by VO's runtime as it should have been.

As innocent as this code looks, there is a problem lurking. To demonstrate the problem consider what would happen if the garbage collector kicked in one line earlier as follows:

```
FUNCTION Start
  LOCAL o AS OBJECT
  LOCAL p AS PTR
  o := Error{}   // some Object
  p := PTR(_CAST, o )
// static memory holds pointer to dynamic type
  ? o, “,”, p  // addresses are the same
  CollectForced()
// moves object, pointer is NOT updated!
  RegisterKID(@p,1,.F.)
// register pointer on object that is not there
  ? o, “,”, p  // addresses are different!!!
  WAIT
RETURN
```

The output from this program is as follows:

```
{(0x0078)0x01040138} CLASS ERROR , 0x01040138
{(0x0078)0x02041124} CLASS ERROR , 0x01040138
Press any key to continue...
```

Notice how the addresses reported on the second line are different and that while the object's address has changed from the first to the second line, the pointer's contents have not. This indicates that the when the garbage collection kicked in due to the call to CollectForced(), the object was moved but the pointer p pointing to the object was not correspondingly updated. This makes sense since the registration of the KID does not take place until after the call to CollectForced(), effectively too late to help in this situation since the pointer being registered is pointing to where the object use to be!

You might be thinking to yourself that this example is quite contrived. And prior to VO 2.5 I would probably agree with you. Most developers wouldn't use CollectForced() in this way. However, with VO 2.5's multi-thread support the potential for this to happen in VO 2.5 multi-threaded applications is very real since the garbage collector (GC) can and will kick in arbitrarily in other threads.

Another way to think about this is as follows. In the example above the CollectForced() call represented the arbitrary triggering of the GC. However, from within a multi-threaded application, it isn't necessary to explicitly call CollectForced()to have this problem occur. Instead all that needs to happen in a multi-threaded application is a thread context switch to occur in the same place where the CollectForced() now appears above (e.g.; between the pointer assignment and the call to RegisterKID()). If a thread context switch does occur at precisely this spot AND the GC kicks in in the other thread (a very real possibility if not an eventuality given the preemptive nature of Win32 operating systems), the problem we witnessed above will in fact occur. This is just one more example of how multi-threaded programming is not for the faint of heart.

Now most of you probably don't have much cause to use the RegisterKID() function in your normal VO business application development. However, VO's internal class libraries use this function all the time. For example, if you own the VO 2.5 version of the SDK, it includes the following source code for the method EventContext:Init(), which is called each time you create any VO GUI Window since VO's Window class inherits from EventContext. The code as it appeared in the 2.5 version of the SDK looked like:

```
method Init() class EventContext
  local strucSelfPtr as SelfPtr
  VTrace VBorder

  super:Init()

  RegisterAxit(self)
  strucSelfPtr := MemAlloc(_SizeOf(SelfPtr))
  strucSelfPtr.ptrSelf := ptr(_cast, self)
// If GC kicks in here we have problems
  RegisterKid(@strucSelfPtr.ptrSelf, 1, false)
  ptrSelfPtr := strucSelfPtr
```

Once again, while the possibility seems remote, we have the potential for big problems with this version of the EventContext:Init() method if the GC kicks in in the indicated spot. While this is virtually impossible in a single threaded application, the possibilities become very real in a multi-threaded application since it is quite conceivable that a thread context switch will eventually occur at precisely the right spot, i.e.; after the pointer strucSelfPtr.ptrSelf has been assigned the address of the dynamic object self, but before the KID gets registered for that pointer.

To be fair, CA officially stated that CAVO2GUI.DLL (the DLL that contains VO's GUI class library), along with most of the other class library DLLs, was NOT thread safe when VO 2.5 was released. This implies that when using the GUI classes, developers should not have two threads of execution occurring at the same time within these classes. However, the above problem can occur even if only one thread is running in the GUI classes, and one or more threads are running in non-GUI code (such as business logic), a common occurrence in multi-threaded VO applications. Again, all that is required to have the problem occur is that a thread context switch occur in the indicated spot, and the GC kick in in that other thread. And when trouble does strike, it is usually miles away in application code with no clue as to where to start looking for the problem – the hallmark of the classical dynamic memory bug. As such I considered this a bug and reported it to CA shortly after VO 2.5 was released.

Fortunately, the solution to this problem is an easy one. Consider the following updated version of my original sample above:

```
FUNCTION Start
  LOCAL o AS OBJECT
  LOCAL p AS PTR
  o := Error{}   // some Object
  RegisterKID(@p,1,.F.)
// register the KID before assigning pointer!!
  p := PTR(_CAST, o )
// static memory holds pointer to dynamic type
  ? o, “,”, p
// addresses are the same
  CollectForced()
// moves object, pointer automatically updated!
  ? o, “,”, p
// addresses are the same!!
  WAIT
RETURN
```

The fix to the problem is to simply ensure you register the KID before you assign the address of the dynamic data type to it. By first registering the KID prior to assigning it a dynamic data type address, you are ensuring that any GC that occurs after the registration and that causes the underlying dynamic data type to move in memory, will automatically cause the address held by the pointer to be updated accordingly. Of course if the GC kicks in even before the address of the dynamic data type has been assigned then no harm is done, and the assignment will simply pick up the new address of the dynamic data type.

Accordingly, the source code to the EventContext:Init() method in the VO 2.5a SDK has been adjusted to include this fix as follows:

```
method Init() class EventContext
  local strucSelfPtr as SelfPtr
  VTrace VBorder
  super:Init()

  RegisterAxit(self)
  strucSelfPtr := MemAlloc(_SizeOf(SelfPtr))
  // Register the KID before
 // the assignment  takes place!!
  RegisterKid(@strucSelfPtr.ptrSelf, 1, false)
  strucSelfPtr.ptrSelf := ptr(_cast, self)
  ptrSelfPtr := strucSelfPtr
  return self
```

This represents a truly thread safe version of the method (considered only in the context of the RegisterKID() call - CA still does not warrant its class libraries as being thread safe in VO 2.5a) since no matter what line of the above code a thread context switch might occur on, the addressed contained in the pointer will always be that of the dynamic object.

So What?

For most of you this article has likely border on next to useless since you are probably not writing multi-threaded applications.

For you, none of what I am talking about explains your particular 5333 errors. However, if you stick with me a little while longer I promise you a just reward.

The above example of a manufactured dynamic memory bug wasn't just made up. It actually came from a real-world situation I was running into when developing a thread-safe 2.5 version of VOCOM, my VO 3rd-party product that allows you to create COM/ActiveX servers of all kinds. In fact, the exact sample above was devised in order to show CA development the problem I was having with respect to threads and my incorrect use of RegisterKID(). What was enlightening to me was that the possibility to incorrectly use this function (i.e.; assign the address of a dynamic memory data type to a static pointer before calling RegisterKID() on that pointer, instead of after), was so easy to do that it struck me that VO's own development team may have made the error themselves. (You have to remember that prior to VO 2.5, VO's runtime was not designed to be thread safe and so any multithreading within a VO application was officially unsupported by CA. Thus, this incorrect use of RegisterKID() never caused a problem since no one was writing multithreaded applications in VO 2.0.) A quick check confirmed several places in various class libraries and in the RDD subsystem where this very incorrect technique was being used and so I promptly reported these hidden time bombs to CA development using the very sample shown above.

However, as I was preparing the bug report to submit to CA something very interesting occurred to me. Consider the following slightly modified version of the original problematic code:

```
STRUCTURE _VOOBJECT
  MEMBER pVTable   AS PTR
  MEMBER pClassInfoAS _VOClass
STRUCTURE _VOCLASS
  MEMBER pNeverMindAS PTR
  MEMBER synClassName  AS SYMBOL

FUNCTION Start
  LOCAL o AS OBJECT
  LOCAL p AS _VOOBJECT
  o := Error{}   // some Object
  p := PTR(_CAST, o )
// static memory holds pointer to dynamic type
  ? o, ",", p, ",", p.pClassInfo.atomClass
// everything is good
  CollectForced()
// moves object, pointer is NOT updated!
  RegisterKID(@p,1,.F.)
// register pointer on object that is not there
  ? o, ",", p, ",", p.pClassInfo.atomClass
// problems!!!
  WAIT
RETURN
```

The output from this program is as follows:

```
{(0x0078)0x01040138} CLASS ERROR , 0x01040138 , ERROR
{(0x0078)0x02041124} CLASS ERROR , 0x01040138 , ERROR
Press any key to continue...
```

Basically, the only thing I have added here is some logic to actually try and use the pointer p which is declared as a pointer to a _VOOBJECT structure which in turn points to that object's _VOCLASS class definition structure (in this case Error's). [Note: See my column in the 04/99 issue of SDT for more information on the undocumented structures _VOCLASS and _VOOBJECT.]

Study the program and its output very carefully. Notice anything strange beyond the obvious problems highlighted earlier? What probably went unnoticed by you, and what makes this particular sample relevant to a discussion of dynamic memory problems such as 5333s in general, is the very fact that this program does not crash! Clearly, the address contained by p does not match that of o in the 2nd line for the reasons give above. However, what was confusing at first to me was why the word "Error" continued to appear at the end of the 2nd line? This value – the result of the expression p.pClassInfo.symClassName – continued to evaluate to Error even when the actual Error object had moved to a new location!! This was a very interesting observation, which gave me real insight into the reason why 5333s where so incredibly difficult to locate.

What's actually going on here is that while the Error object has indeed moved to a new location, the original memory location it use to reside at (and to which p still points), while having been freed so that the memory can be reused, has not been zeroed! This means that until the memory does get reused, a memory image of the Error object will remain at that position in memory. This is why the expression p.pClassInfo.symClassName still reports Error since the memory at that address still holds a complete image of the Error object as it was prior to it moving.

If you think about this for a moment you start to catch a glimpse of the debugging horrors this can cause. Instead of crashing when accessing this stale object, the program continues on oblivious that anything has gone wrong. If you extrapolate this example out a little, and imagine a more interesting object than Error, where the object's instance variables are being updated, the above scenario would report the stale (pre-relocation) instance variable values causing you to wonder if you have gone mad. "I just updated the instance variables of the object above and yet the debugger is showing me I didn't! What the #$%# is going on?".

Even worse, is that BOOM all of a sudden an object you have been working with all along (which is actually stale although you don't know it) just simply disappears on you. "There it is in the debugger on line 101, and when I single step to line 102, POOF! it is gone!!" Sound familiar? The reason this can happen is that stale memory that you are actually reading has, if you recall, been released and is available for reuse by the VO runtime's dynamic memory manager. At any point that dynamic memory is required the system can and will overwrite that memory with new unrelated data causing the data you where erroneously working with to simply get blown away. I have even seen the case where one stale object gets complete (and perfectly) overwritten with a new object of a different class. The result was that the debugger was showing that variable o was an object of class "SomeClass" on line 250 and then an object of class "SomeOtherClass" on line 251 after a simple single step!! Of course, running the program outside of the IDE or even within the IDE but not under the debugger, causes GC to kick in at a slightly different time and the problem will not show up at that point (if at all). Oh the joys of dynamic memory problems.... It is enough to make you want to give up software development and become a pastry chef!

WipeDynSpace To the Rescue!

To get right down to the essence of the problem, it is the fact that above program doesn't crash on us when the stale memory pointed to by p is accessed that creates the problem in the first place. If only there was someway to get VO to raise an error immediately when this stale memory is accessed, we would be able to bring the symptom of the problem very much closer to its cause, and therefore make locating dynamic memory bugs infinitely easier to locate.

As I observed the above phenomenon it occurred to me that all of these problem "masking" issues would go away if only VO zeroed the dynamic memory it released. This would then cause any program that accessed stale memory to error sooner since presumable it would not be expecting the memory to be zeroed. Obviously, there would be a price to pay for this "feature" since it would actually slow down the application slightly to have to zero all dynamic memory that is released by the system. However, I reasoned that the benefit zeroing memory would have in terms of being able to more easily debug the application could outweigh any downside to performance in the application. I think most of us would accept our application running ever so slightly slower, if in return we could get rid of our 5333s. Fortunately, the obvious solution of a dynamic runtime registry setting occurred to me which would allow developers to debug their software in a slightly slower environment whereby released memory is being zeroed, without slowing down the final delivered production system on a client's machine. I therefore made a request that VO development add an undocumented registry setting that will cause VO's runtime to write CHR(0)'s to all dynamic memory the moment it is released. I am happy to report that CA development accepted my proposal and that beginning with VO 2.5a the following registry setting will control whether VO's runtime will zero released dynamic memory or not.

```
REGEDIT4

[HKEY_CURRENT_USER\Software\ComputerAssociates\C
A-Visual Objects Applications\Runtime]
"WipeDynSpace"=dword:00000000
```

The default as shown is 0x00000000, which represents the current VO behavior not to zero released memory. By simply adding the above key to your registry (its not there by default) and setting the value instead to 0x00000001 all dynamic memory will be automatically zeroed by the VO runtime upon release.

If you do this and run the above problematic code again, you get the satisfying result of a GPF on the 2nd line since the expression p.pClassInfo.symClassName will cause a null pointer to be dereferenced.

I tell you quite truthfully, I was never more happy to see a GPF in all my life then when I ran the above program with the new registry setting in place! With the GPF result, identifying the underlying problem in this case becomes straight forward.

Conclusion

If you have been traumatized by incessant dynamic memory errors of the 5333 variety, then VO 2.5a offers real hope for you. Simply set the above registry setting to 0x00000001 permanently on your development machine and start seeing some of those 5333 errors begin to reveal themselves to you much closer to their point of origin. We all know that if you can reliably reproduce a problem then you can generally readily fix it. This single registry setting, for me, is without doubt the most important addition to VO 2.5a since it brings to my attention dynamic memory problems much sooner then they otherwise would be (if at all). The feature will go along way to stabilizing a great many VO applications currently begin bitten by dynamic memory problems.

If you have been thinking about moving back to VO 2.0 rather than proceeding to VO 2.5a because your confidence in VO 2.5 has been shaken by a recent a rash of 5333s, I recommend giving 2.5a a try and seeing if this new undocumented registry setting doesn't help you solve some of your problems. While its true that you still have some debugging to do to find the actual cause of your dynamic memory problems, at least with this new registry setting you have a fighting chance. Also, if you are trying to write a multi-threaded VO application in VO 2.5, I strongly recommend moving to VO 2.5a for the many bug fixes it contains with respect to its internal use of the RegisterKID() function as mentioned above.

Until next time…

Dieter Crispien

Cool New Menus - Part 2

In part 1 (see SDT 1/99), I showed how menu entries are created which contain bit-maps instead of text. In this issue, I want to demonstrate how one can display both text and bitmaps in a menu entry. One is then fully prepared for the fashionable innovations, which Office97 has made familiar.

Nothing works without ownerDraw

As it often comes, if one wants to deviate from the standard, MicroSoft enables us to take control of the symbol work oursel-ves. This can be found in the MSDN Library via the keyword *MFT_OWNERDRAW*. In order to assign this attribute to a MenuItem, we create a sub-class of MenuItem, which we use after the window, and its menu have been created.

```
METHOD Init(hMenuPopup,nID,lHasBitmap) CLASS
CoolMenuItem

  LOCAL sMii IS _winMenuItemInfo
  LOCAL pMyItem AS MyItem
  Default(@lHasBitmap,FALSE)
  SELF:lHasBitmap := lHasBitmap
  IF hMenuPopup == NULL_PTR
    // Error
  ELSE
    GetMenuItemInfo(;
     hMenuPopup,nID,FALSE,@sMii)
    SELF:dwMenuID := nID
    SELF:dwStatus := sMii.fstate
    SELF:dwType := sMii.fType
    IF SELF:lHasBitmap
     pMyItem := ;
     MemAlloc(_sizeof(MyItem))
     MemClear(pMyItem,;
     _sizeof(MyItem))
     // change the item to an owner-drawn item
    // and save the address of the
     // item structure as item data
     sMii.fMask := MIIM_TYPE //_Or(MIIM_TYPE,MIIM_DATA)
     sMii.fType := ;
     MFT_OWNERDRAW
     SELF:dwType := sMii.fType
     sMii.dwItemdata := ;
     DWORD(_CAST,pMyItem)
     SetMenuItemInfo(;
     hMenuPopup, nID, FALSE,;
      @sMii)
    ENDIF

  ENDIF
```

When manual drawing comes into play later, it is crucial to store the information given to the user-defined structure pMyItem. This is done via the *dwItemData* member, as can be seen above. With the *Init* method shown above, it is essential that it is called at the end of the initialization phase of a window. Therefore we intercept the following message in the Dispatch() of the window, which owns the menu:

```
METHOD Dispatch(oEvent) CLASS dwCustomer
     LOCAL dwMsg AS DWORD
     dwMsg := oEvent:Message
     DO CASE
     CASE (dwMsg == WM_PARENTNOTIFY)
     IF LoWord(oEvent:wParam)= WM_CREATE
        SUPER:Dispatch(oEvent)
        SELF:ChangeCoolMenu(oEvent)
        RETURN 0L
     ENDIF
     // any extensions go here
```

And in *ChangeCoolMenu()*, we replace the menu items with our own CoolMenuItems.

```
METHOD ChangeCoolMenu(oEvent) CLASS dwCustomer
IF SELF:menu != NULL_OBJECT .and. ;
     IsMethod(SELF:menu,#ChangeCoolMenu)
     SELF:menu:;
     ChangeCoolMenu(oEvent,SELF)
ENDIF
RETURN 0L
```

Our new Coolmenu class (inherited from the menu) then adjusts the desired *MenuItems*:

```
METHOD ChangeCoolMenu(oEvent,oWin)CLASS CoolMenu
LOCAL sMii IS _winMenuItemInfo
LOCAL hMenubar AS PTR
LOCAL hMenuPopup AS PTR
LOCAL oItem AS CoolMenuItem
LOCAL nID AS DWORD

hMenubar := SELF:handle()
DO CASE
CASE IsInstanceOf(oWin,#DataWindow)
  GetMenuItemInfo(hMenuBar,;
  IDM_StandardShellMenu_File_ID,FALSE,@sMii)
  hMenuPopup := sMii.hSubMenu
  FOR nID := IDM_StandardShellMenu_File_Open_ID;
      UPTO IDM_StandardShellMenu_File_Exit_ID
    oItem := ;
    CoolMenuItem{hMenuPopup,;
    nID,TRUE}
    AAdd(SELF:aCoolItem,oItem)
  NEXT nID
```

For all of the items we have set to MFT_OWNERDRAWn, we need to respond to the WM_MEASUREITEM and WM_DRA-WITEM messages. When other routines need to reset the ownerdraw attribute, this can also be done when the message WM_INITMENUPOPUP is processed.

Linking ToolbarButtons with our MenuItems

The Menu Editor connects ToolbarButtons to MenuItems. These ToolbarButtons are taken from a bitmap ribbon, either from VO's standard ribbon or self-drawn ribbon. For our MenuItems, however, we need these as ImageList items. Therefore we make it as simple as for ourselves, and stuff the existing ToolBar into an ImageList.

```
// extract from an extended
// ChangeCoolMenu-Method of the CoolMenu-
// Class:
oTB := oWin:Toolbar
oImageList := ImageList{255,{20,16}}
oImageList:Add(oTB:Bitmap)
SELF:oImages := oImageList
```

The VO Toolbar can hold up to 255 pictures which are each 16x20 (see *AppendItem* method in CAVO25.Hlp). The MSDN Library also shows us, that we can add several images to an ImageList at the same time. The number of pictures is determined from the total width of the Toolbar bitmap divided by the size of the individual bitmaps, as described in the help file under *ImageList:Init()*.

One more extension to *CoolMenu's ChangeCoolMenu* method adds the index for each Item to the appropriate image:

```
nBID:= AScan(oTB:aTipsText,;
{|x|x[2]=nID})
nButtonID := oTB:aTipsText[nBID,1]
IF nButtonID > 0
  AAdd(SELF:aToolbarId,;
{nID,nButtonID})
  oItem:nButtonIndex := nButtonID
ELSE
  oItem:nButtonIndex := -1
ENDIF
oItem:dwMenuID := nID
```

Since VO already does this allocation in the array aTipsText, we profit from this by writing an access. Then, our Coolmenu object contains all necessary information for drawing the bitmaps.

Measuring MenuItems

Before MenuItems are drawn, they are measured. The owner window receives a message WM_MEASUREITEM and reroutes that to the appropriate event handler, where we fill a structure with the information needed for drawing the MenuItems. *DrawText()* (with the DT_CALCRECT parameter) is the essential function used by the MeasureItem event handler. This calculates the size of the text. We need to pad some space for the edges (CXTEXTMARGIN), and also for the width of the button image (oButtonSize:width), and for some distance between the button and the menu text (CXGAP). In every case, one should get the item dimensions from the system settings via *GetSystemMetrics()* (SM_CYMENU gets the menu height. SM_CXMENUCHECK gets the width of the standard chekkmark, which we will replace with our bitmap, and therefore must subtract from our total width).

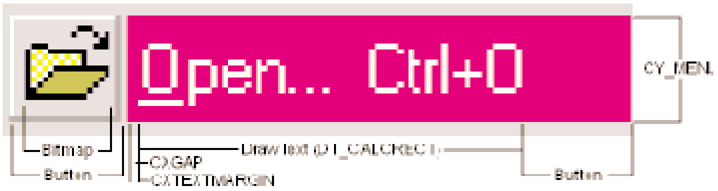


Abb. 1: Dimensions of a MenuItem

Details can be gathered from the figure 1, which I took from the article of Paul DiLascia [2].

This information is stored in the structure *_winMEASUREITEMSTRUCT*, which is defined by VO. In order to guarantee that we properly fill this structure for the Drawtype menu, we need to test if the CtlType member of this structure is ODT_MENU at the beginning of our MeasureItem routine. The separators require special handling, since they use only half of the height of a standard Item.

Drawing a MenuItem

After all these painstaking preparations, the MenuItem can be drawn. After Windows measures the Item to be drawn with the help of our MeasureItem routine, it sends a WM_DRAWITEM message to our window, one for each Item. Equipped with the information in _winDRAWITEMSTRUT, we can really go wild with this. Colours and 3D edges (raised or pressed) depend on the itemstate (ODS_GRAYED, ODS_SELECTED, or ODS_CHECKED). It is also crucial here whether a symbol should be drawn. Or perhaps an user-defined check mark comes into play? It doesn't matter which symbol, it comes back to its position as represented in fig. 1

The great pain of ownerdrawn programming

Windows is extremely hard on programmers, who have taken the trouble to program ownerdraw. Not that everything now works. We still need to re-invent the functionality of the underlined letters. The underlinings are displayed, but we need an additional event handler, which reacts to WM_MENUCHAR. This example still contains no code for that. This may be delivered subsequently with the next issue of SDT. By the way, accelerators are actually not affected by the ownerdraw attribute, which means, they work normally.

The necessary cleanup work

Since we define our own structure for extending the standard MenuItemInfo structure, we need to release these pointers upon closing the window.

Colors and fonts

Often the system settings for colors and fonts are forgotten by commercial applications. Paul DiLascia's article [2] recommends using the functions *GetSysColor()* and *GetSystemMetrics()* against the eventuality that the user chooses their own colors and fonts. Additionally, one should react to the messages WM_SYSCOLORCHANGE and WM_SETTINGCHANGE. The most reliable reaction is still to destroy everything and build it back up from scratch. This table lists the most important parameters for *GetSystemMetrics()* and *GetsysColor()*, as far as menus are concerned:

COLOR_MENU	Menu color
COLOR_MENUTEXT	Text color in Menu-Item
COLOR_HIGHLIGHT	Background color for selected Menu-Item
COLOR_HIGHLIGHTTEXT	Text color for selected Menu-Item
SM_CYMENU	Height of a Menu-Item
SM_CXMENUCHECK	Width of a Menu Checkmark

Closing remarks

Many additional problems related to creating ownerdrawn menus are covered by Paul DiLascia [2]and go beyond the limits of an introductory article, such as this.

Who now believes that Office97 and Office2000 are all working with ownerdraw menus, is wrong. For this purpose, Microsoft created something new: CommandBars. The object model for these COM Objects is documented in the VBAOff8.HLP. Unfortunately, these objects can only be used in the Office applications themselves.

After going through all of the trouble to develop this VO example, I can easily understand why developers want to stay away from owner drawn menus, and wait for Microsoft to include CommandBars in the ComCtl32.Dll. There is, naturally, no official promise that this will ever happen.

Nevertheless, I dared dealing with a further customdraw example and wrote a TreeListView for Cayman, which will be introduced in the next SDT issue, with versions for VO and for ClassMate.

Literature

[1] MSDN Library, see under Platform SDK/User Interface Services/Windows User Interface/Resources/Menus/Using Menus/Creating Owner-Draw Menu Items

[2] “Give Your Applications the Hot New Interface Look with Cool Menu Buttons” by Paul DiLascia, MS System Journal 1/98.

An excellent article, well worth reading, concerning the troubles of ownerdraw programming. How much the emotions can be involved, is showed by the following short quotations:

“Too much Windows programming isn´t healthy, you know”
“... a waste of precious neurons”
“... until my fingers were in danger of carpal collapse.”

Dieter Crispien

Email: DCrispien@dcsoft.de
CompuServe: 100016,1673

From relational to object-oriented data structures

Michael Zech

Since most developers today must consider existing data, converting relational data structures to an object-oriented approach has significant meaning for those converting to Jasmine. This article discusses conversion considerations, beginning with the well-known and loved DBF files. Also, the Jasmine Script Utility will be introduced, as a conversion aid. The utility can be found, with source, on the accompanying CD.

Introduction

One of the many impressions which I brought home with me from numerous DevCon99 discussions, was the realization that VO developers (who still use predominately xBase files for databases), in particular, are interested in Jasmine. Even if the advantages of a contiguous object oriented approach, both in the programming environment, as well as with the data structures is approved, some can't imagine, how this can be realized in practice.

For this reason, I want to point out in this article, using the simplest steps possible, how to convert existing relational data structures into object-oriented ones.

Preparations

In order to present a better overview, beginning with familiar xBase tables seems best. However, in order to be able to demonstrate the migration as descriptively as possible with a practical example, a few conditions need fulfilling however:

- First an existing data structure is needed, which uses a typical relation. We find this in the \CAVO25\Samples\GSTutor directory.

- On the object-oriented side, we need a OO database. In addition to CA-Visual Objects 2.5a, Jasmine should be installed. The free developer edition of Jasmine 1.21, or the freely available beta of Jasmine ii from http://www.ca.com, will work nicely for this.

- A Microsoft VC++ compiler (5.0 or better) is needed, since Jasmine uses this for compiling the classes. This is a system requirement for Jasmine.

- Finally, we need a text editor, such as Notepad, which can create and edit ODQL scripts.

Now, since we have all of the necessary tools, we can begin. Since our relational data structure will be transmitted to the Jasmine database server, a place for the object-oriented structures must first be determined.

Here Jasmine serves up a characteristic, the ClassFamilies, which I have already described in detail in the article "VO meets Jasmine". Despite that, a cursory description of the meaning of class families is in order, and of how one builds them.

A ClassFamily is a grouping of classes sharing, if possible, a logical context. However, this is not a compelling pre-requisite. Assuming that a comprehensive business model represents data for purchases, sales, inventory, and personnel, then a ClassFamily could be created for each of these groups, and all data classes are combined according to their respective affiliations.

The principle that every class belongs to a ClassFamily is not optional. As we will see in the ODQL Scripts, most instructions involving classes also require the ClassFamily name. On the other hand, belonging to a ClassFamily does not any affect the inheritance hierarchy. ClassFamilies are **not** super classes!

We will create a ClassFamily with the name *SDTSamplesCF*, where we can store the converted classes. From DOS, we give Jasmine the command *CreateStore*. Since it is possible to capture such instructions in a batch file under Windows NT, we will. Since Jasmine is extremely case sensitive, batch files are particularly useful for avoiding typos.

```
@Echo Off
rem ======================
rem SDTSamplesCF .CMD
rem ======================

CLS

Echo Creating class familiy...
CreateCF SDTSamplesCF dataStore
```

The opposite, to remove ClassFamilies:

```
@Echo Off
rem================================
rem SDTSamplesCF.CMD

CLS

Echo Delete class familiy...
deleteCF SDTSamplesCF dataStore
```

Jasmine *ii* offers additional classes and methods for these administration functions, which enables the execution of ClassFamilies for example by VO applications. The approach with the command line, as shown above, however remains valid, and is also better suited for the understanding the current topic.

From the table to the object

When it comes to the creation of table structures, VO developers, who still trust the venerable dbf format, are quite spoiled. This is usually done either by exporting from VO's dbServer editor, or at run-time from the application. A raft of efficient methods and functions are available for this. One does not have to worry about any further aspects of security or administration, as DBF tables never had these anyway. This comportment results from the peculiarities in the early days of the PC, where the xBase file system originated.

With inveterate SQl fans, this comportment would cause at least a frown, but more likely sheer horror, because relational database systems (such as Ingres, or MS SQL Server) behave completely differently in this regard. They have a much different system of rights. And, furthermore, it is not customary for an application to alter SQL database structures at runtime. The creation of the data structure is a one event for these systems, which starts with the design phase. Or at least it should.... After finishing the design phase, scripts are made, with which the database is automatically set up. SQL serves as the language for these scripts for relational database management systems (RDBMS). Since this procedure makes for a chunk of manual work, some visual design (or CASE) tools (Rational Rose, S-Designer, etc.) are available.

Jasmine's behaviour, as an object-oriented database, has more in common with SQL than with DBF. Altough the data structure consists of a class hierarchy rather than RDBMS structures, the installation is still script driven. As a language, ODQL functions as the object-oriented counterpart to SQL.

The relational basis

As suggested before, we now fall back to the file *customer.dbf* for our conversion. The data structure is represented as follows:

Column	Type	Lenght	Decimals
Custnum	N	5	0
Firstname	C	10	0
Lastname	C	10	0
Address	C	25	0
City	C	15	0
State	C	2	0
Zip	C	5	0
Phone	C	13	0
fax	C	13	0

As the following mini-program shows, the data structure of a dbf file can comfortably be selected with few lines of VO code.

```
FUNCTION Start()
LOCAL aStruct AS ARRAY
LOCAL oServer AS dbserver
LOCAL i AS DWORD

oServer := dbsever{“ customer.dbf”}
aStruct := oServer:DbStruct
FOR i := 1 TO Len(aStruct)
    AEval(aStruct[i],{|x|QQOut(AsString(x)+ ;
   Replicate(“ “, 10-Len(AllTrim(AsString(x)))))})
    QOut(CRLF)
NEXT

WAIT
```

Converting the data types

As always, when data is transferred between systems, the question is which data types will the new system support? Converting to the data types of the target system is a requirement. In our case, this remains the same, as the following comparison of the respective data types shows:

	Dbf	Jasmine
GanzZahlen	numerisch	Integer
Fließkomma	Decimal	Decimal, real
Text	Character	String
Datum	Date	Date
Logisch	Logic	Boolean
Binär	Memo	ByteSequence
Objekte	-	Object

Pulling out the documentation, the trained eye immediately sees that there are a couple of changes to be made. For the pure conversion of the data structure these differences are not particularly large. However, if the data cannot be transferred in one batch, the data types need to be well chosen. Fortunately, CA-Visual Objects has an abundance of functions for converting various data, so that it conforms with Jasmine's needs. Describing such a conversion easily offers sufficient material for an additional article, therefore we will assume that no data conversion is needed at the moment, and turn our attention to script writing, which will create classes to represent our relational table.

Designing That First Class

In order to clarify the difference between relational tables and object-oriented classes, I consciously selected a very simple file: *Customer.dbf*.

At first glance class design could look quite complicated, however the details prove to be quite simple. Relational databases consist of tables, which are divided into fields and records. The fields describe the characteristics of the data stored in the records.

Where the relational approach uses tables, the object-oriented approach naturally uses classes, which possess properties. These properties correspond to the fields of a table. The actual data is stored in the objects, which instantiate the class. For the current example, the first class design could look like:

Class:	CUSTOMER
Properties:	CUSTNUM
	FIRSTNAME
	LASTNAME
	ADDRESS
	CITY
	STATE
	ZIP
	PHONE
	FAX

The first script

As before hinted, Jasmine speaks its own language: ODQL. The abbreviation stands for **O**bject **D**ata **Q**uery **L**anguage, a query language for objects. The language syntax uses uses a mixture of SQL and C++.

One of the first important characteristics is case sensitivity. Pure VO developers need to accustom themselves to it from the beginning, before disrespecting the case creates nasty problems.

The requirement of ending each command with a semicolon (;) comes from C. This also true of blocking statements with curly braces, in order connect them. As with C, type definitions always appear before variable names. Roughly simplified the following syntax pattern emerges:

```
ODQL Syntax:

1. command;

/* Comment */
2. command
{
more commands
more commands
...;
};
3. command;
...
..
```

Now, in order to build that script, we need some ODQL instructions. First of all, Jasmine needs to be told to which ClassFamily the new class belongs. Our example has already done that, and we can now use this ClassFamily. ODQL also has the *defaultCF* command for this purpose, which could well be one of the most frequently used instructions.

In order to create the class, we use the instruction *defineClass*, followed by the class name. The ClassFamily can be explicitly declared by the syntax <ClassFamilyName>::<ClassName>, thereby avoiding the *defaultClass* command. Note that the proper sign is two colons. However, if this is done, then it must be carried through throughout the entire script.

The instruction *defineClass* wants three parameters altogether. The first, when inheritance is desired, indicates the name of the superclass. The second parameter optionally provides the class with a description. The third parameter consists of a block of instructions, which determine the necessary properties of the class with their respective properties. Two more keywords determine whether the characteristics are valid at class or object level.

Jasmine distinguishes between properties applying to all objects of a class and those applicable to only to each instance of the class. For our example, only characteristics applying to instance levels make sense, and that gives the following script:

```
defaultCF SDTSamplesCF;

/* Define a new class */
defineClass CUSTOMER

description: “”

{
    instance:
        systemCF::Integer     CUSTNUM;
        systemCF::String[10]  FIRSTNAME     ;
        systemCF::String[10]  LASTNAME     ;
        systemCF::String[25]  ADDRESS     ;
        systemCF::String[15]  CITY     ;
        systemCF::String[2]   STATE     ;
        systemCF::String[5]   ZIP     ;
        systemCF::String[13]  PHONE     ;
        systemCF::String[13]  FAX     ;

};
buildClass CUSTOMER;
```

The final command *buildClass* tells Jasmine to create the new class. This process creates the pre-processor code, and passes them to the C-compiler. There is the reason for the aforementioned compiler requirement, which must be installed on the same machine as the Jasmine database.

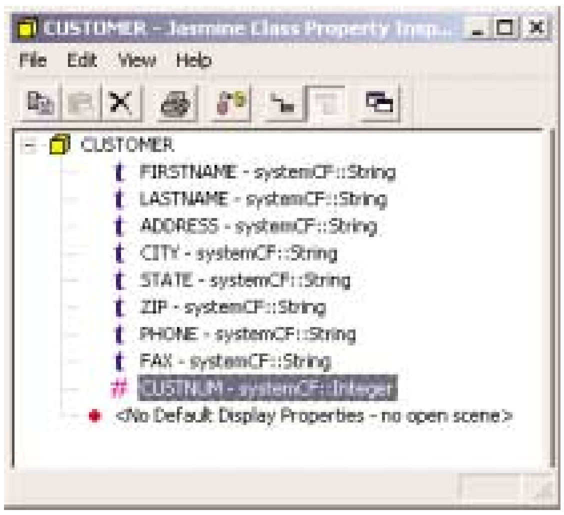
Now we only need store our script to an .ODQ file and execute it. Therefore we use the command line interpreter CODQLIE with its parameter –exeFile (caution: case sensitivity!).

C:>codqlie –execFile customer.odq

The successful storing and building of the class is confirmed by the following interpreter responses:

```
E:\ODQ-Files>codqlie -execFile customer.odq
Client ODQL Interpreter
Jasmine Version 2.0
 Portions of this product Copyright 1996-1999
Computer Associates International, Inc.
 Portions of this product Copyright 1996-1999
FUJITSU LIMITED
 Portions of this product Copyright 1996-1999
Computer Associates International,
 Inc. & FUJITSU LIMITED

Connecting to host ZECMI02.
(Information) E_OD6297_ODB_ECE_OKDEFCLASS Class (
‘SDTSamplesCF’::’CUSTOMER’ )
has been successfully defined.
(Information) E_OD6294_ODB_ECE_OKBLDCLS Class (
‘SDTSamplesCF’::’CUSTOMER’ )
has been constructed successfully.

E:\ODQ-Files>
```

“Faith is good, control is better!” For this reason, we can also check the results via Jasmine Studio:



```
CUSTOMER
    FIRSTNAME - systemCF::String
    LASTNAME - systemCF::String
    ADDRESS - systemCF::String
    CITY - systemCF::String
    STATE - systemCF::String
    ZIP - systemCF::String
    PHONE - systemCF::String
    FAX - systemCF::String
    CUSTNUM - systemCF::Integer
    <No Default Display Properties - no open scene>
```

Relationships

Real life doesn't consist of individual tables. The name **relational** database points to the reality that more of the story is told through the relationships between the tables. Only the proper usage of these relationships avoids redundancy. In most cases, the relationships connect only a few tables. However, if the data structure represents more complex business procedures, relational structures can break down quickly and then now amount of programming will rein in these databases. In such situations a representation of the data in a class hierarchy has clear advantages.

So, how does the typical 1:N relationship look in a class hierarchy? Let's take our class *CUSTOMER*, which we just built, and relate it to another table: orders. The example directory GSTutor of CA-Visual Objects again provides us with an appropriate example table, *orders.dbf*. We first have to build it as a class in Jasmine, just as we did with the customer table. The folloing script does that. With our previous experience, it should not need further explanation:

```
defaultCF SDTSamplesCF;

/* Define a new class */
defineClass ORDERS

description: “class for customer orders”

{

    instance:
        systemCF::Integer       CUSTNUM     ;
        systemCF::Integer[5]    ORDERNUM     ;
        systemCF::Date[8]       ORDER_DATE     ;
        systemCF::Date[8]       SHIP_DATE     ;
        systemCF::String[25]    SHIP_ADDRS     ;
        systemCF::String[15]    SHIP_CITY     ;
        systemCF::String[2]     SHIP_STATE     ;
        systemCF::String[5]     SHIP_ZIP     ;
        systemCF::Decimal[10,2] ORDERPRICE     ;
        systemCF::String[5]     SELLER_ID     ;

};
buildClass ORDERS;
```

In order to relate these two tables, we need at least one identical column per table.

In the object-oriented approach we simply extend *CUSTOMER*, which serves as the master, with one more property. This property can now be an object of the class *ORDERS* or through a method, supply us an *ORDERS* collection.

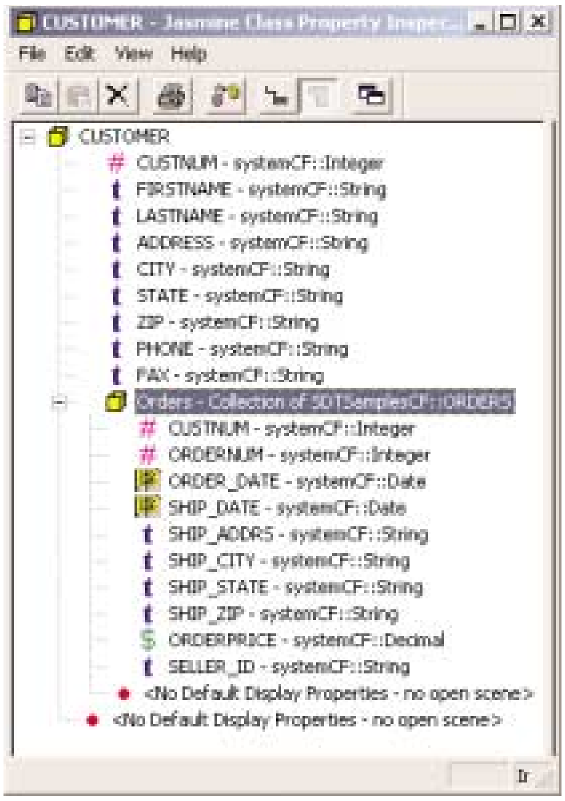
In the first case, we have two options. If the property contains an individual object at run-time, then we have a 1:1 relation. If we want to access several orders, we need a 1:n relation. In this case the property should contain a collection. The typical collection type for this is a Bag. To make this possible, we have to add the appropriate instruction to the script customer.odq.

Bag<SDTSamplesCF::ORDERS>   Orders   ;

Important: As with many RDBMS, Jasmine also does not permit dynamic changes to a class hirarchy, for security reasons. That means, when changes are made, the existing structure has to be changed and then be built again with the following script:

```
defaultCF SDTSamplesCF;
/* Define a new class */
defineClass SDTSamplesCF::CUSTOMER
description: “”
{
    instance:
        systemCF::Integer    CUSTNUM    ;
        systemCF::String[10] FIRSTNAME    ;
        systemCF::String[10] LASTNAME    ;
        systemCF::String[25] ADDRESS    ;
        systemCF::String[15] CITY    ;
        systemCF::String[2]  STATE    ;
        systemCF::String[5]  ZIP    ;
        systemCF::String[13] PHONE    ;
        systemCF::String[13] FAX    ;
        Bag<SDTSamplesCF::ORDERS> Orders    ;
};
buildClass CUSTOMER;
```

What really happened becomes evident, when the *CUSTOMER* class is checked via the Jasmine Class Property Inspector:



```
CUSTOMER
    CUSTNUM - systemCF::Integer
    FIRSTNAME - systemCF::String
    LASTNAME - systemCF::String
    ADDRESS - systemCF::String
    CITY - systemCF::String
    STATE - systemCF::String
    ZIP - systemCF::String
    PHONE - systemCF::String
    FAX - systemCF::String
    Orders - Collection of SDTSamplesCF::ORDERS
        CUSTNUM - systemCF::Integer
        ORDERNUM - systemCF::Integer
        ORDER_DATE - systemCF::Date
        SHIP_DATE - systemCF::Date
        SHIP_ADDRS - systemCF::String
        SHIP_CITY - systemCF::String
        SHIP_STATE - systemCF::String
        SHIP_ZIP - systemCF::String
        ORDERPRICE - systemCF::Decimal
        SELLER_ID - systemCF::String
        <No Default Display Properties - no open scene>
    <No Default Display Properties - no open scene>
```

The newly added property *Orders*, with all its associated properties, is linked directly with the class *CUSTOMER*. If I would like to access the orders in my VO application later, I can do this directly via *CUSTOMER*, as one can see in the next code fragment:

```
LOCAL oCustomer AS JObject
LOCAL oOrders AS Jcollection

…
oCustomer := JObject{….}
oOrders := oCustomer:Orders
```

The two last lines deliver a collection of *ORDER* objects for processing. Even if this representation is heavily simplified, it nevertheless shows, how consistently object-oriented data is handled by an OO-language such as CA-Visual Objects. One good, clear, example of how to access Jasmine data is the Jasmine Simple sample from CA- Visual Objects.

Parents have children

One of the fundamental reasons for a relational database is avoiding data redundancies, because the information necessary is distributed between different tables. Thus for example the address information is put down in a separate table, which can be connected if necessary with both the customers and suppliers tables by a key field. In plain language, that means that the specific person only gets the necessary data associated with the customer, when accessing the customer table, or the supplier associated information from the supplier table. Data, which involves both tables, is stored in a separate table. The disadvantage of this form of data design is that as the data structures increase, the necessary number of the relationally connected tables rises so quickly that the clarity can be lost.

The inheritance methods of the object-oriented approach offers in comparison clear advantages.

Thus, with inheritance, a base object or a superclass, which contains all common information, must first be created. Under this superclass almost as many sub-classes can be created as desired, all of which have access to the information of the superclass.

If one transfers this to the example described above, we would have different groups of persons in a fictitious enterprise: customer, supplier, and coworker. These were then summarized in a superclass *people*. This superclass then contains characteristics, which are common to all inherited classes.

To test this inheritance in the context of a practical example, we start from the following inheritance tree:

```
SDTSamplesCF
 ├─ Person
 │   │
 │   ├─ Customer
 │   ├─ Supplier
 │   └─ Employee
 ├─ Orders
 ├─ BankInfo
 └─ Articles
```

This means that now the class *people* will contain the information, which the sub-classes also need; thus Name, Address, und Contact Data. Specialized data regarding orders, articles, or accounts, are then stored in the sub-classes.

It would be sensible to consider order in which classes are generated, before creating the necessary scripts. The reason for this is that if a class depends on a second class, the second class needs to exist in the database, before the first class can be created, otherwise class generation fails. Since *Orders* already exists and does not require any changes, we can create the first script for the *Aticle* and *Bank* classes in the defaultCF SDTSamplesCF:

```
defaultCF SDTSamplesCF;
/* Define a new class */

defineClass SDTSamplesCF::ARTICLES
description: "Superclass for articles"

{

  instance:
    systemCF::String[20]    ARTNR    mandatory:;
    systemCF::String[80]    ARTNAME    mandtory:;
    systemCF::String    ARTDESCR    ;
    mediaCF::CABitmap    PICTURE    ;

};
buildClass ARTICLES;

defaultCF SDTSamplesCF;

/* Define a new class */
defineClass SDTSamplesCF::Bank
description: "superclass for bank related information"
{
    instance:
      systemCF::String[20] blz  mandatory:;
      systemCF::String[80] name mandatory:;
      systemCF::String[30] account mandatory:;
};
buildClass Bank;
```

This new script uses a new keyword: *mandatory*, which means that when instances of the class are saved, the value must be set as well.

We come now to the Script, which creates the class *Person*, as well as its sub-classes.

```
defaultCF SDTSamplesCF;

/* Define a new class */
defineClass SDTSamplesCF::Person
description: "Superclass for all business persons"
{

    instance:
        systemCF::String[40]    FIRSTNAME    ;
        systemCF::String[40]    LASTNAME    ;
        systemCF::String[80]    ADDRESS    ;
        systemCF::String[40]    CITY    ;
        systemCF::String[40]    STATE    ;
        systemCF::String[10]    ZIP    ;
        systemCF::String[20]    PHONE    ;
        systemCF::String[20]    FAX    ;
};
buildClass Person;
```

This class corresponds essentially to the *Customer* class from our first example.

```
defaultCF SDTSamplesCF;

/* Define a new class */
defineClass SDTSamplesCF::CUSTOMER
description: "class inherit person"
super: SDTSamplesCF::Person

{

    instance:
        systemCF::Integer    CUSTNUM    ;
        SDTSamplesCF::Bank    BANK    ;
        Bag<SDTSamplesCF::ORDERS>    ORDERS    ;
};
buildClass CUSTOMER;
```

```
defaultCF SDTSamplesCF;

/* Define a new class */
defineClass SDTSamplesCF::SUPPLIER
description: “class inherit from person”
super: SDTSamplesCF::Person

{

    instance:
        systemCF::Integer    SUPPNUM    ;
        SDTSamplesCF::Bank    BANK    ;
        Bag<SDTSamplesCF::ARTICLES>    ARTICLES
mandatory:;

};
buildClass SUPPLIER;

defaultCF SDTSamplesCF;
/* Define a new class */
defineClass SDTSamplesCF::EMPLOYEE
description: “class inherit from person”
super: SDTSamplesCF::Person
{

    instance:
        systemCF::Integer    PERSNUM    mandatory:;
        SDTSamplesCF::Bank    BANK    ;
        systemCF::Decimal[8,2]    SALARY    ;

};
buildClass EMPLOYEE;
```

Now we have another new keyword: *super*, which defines the parent class. It is still worth mentioning that the properties for the relations to *ORDERS* and *ARTICLES* are of the data type *Bag*, and later contain of Collections of the appropriate objects.

Before you implement the scripts with CODQLIE, you should check if the class *Customer*, which we created in our first example, still exists, and delete it, if necessary. If this is not done, Jasmine generates an error message and all remaining instructions are cancelled. There is no reason to fear that the server, or the database, could be damaged.

Result

I hope you now have an impression of the possibilities offered by Jasmine for converting existing relational data structures to object-oriented methodology. Whereby I would like to stress especially that it is not the goal of Jasmine to replace all relational databases. Rather Jasmine databases should normally complement, and merge the existing volume of data. This becomes particulary clear with the concept pursued by Jasmine *ii*. The so-called *Intelligent Infrastructure* (indicated by *“ii”)*, offers the integration of almost all data sources, whether they now come from Mainframe, Unix, or PC systems.

Since this article was meant as a beginning I will continue this topic loosely in the following issues. Among other things I will tell you how you can merge “Serverside Methods” into your data structure in order to unburden client applications and your network. If you would like to test the construction of classes in the meantime, become familiar with the appropriate ODQL instructions. In Jasmine’s online documentation, which can counted among the few successful reference texts, you will find these instructions described in detail. As aids to generating scripts, you can use the Jasmine Script Utility, which is described in the article of the same name in this issue. This tool permits you to interactively create ODQL scripts based on template files.

Michael Zech
Email: mz@michaelzech.de

Jasmine Script Utility

Michael Zech

The Jasmine Script Utility is intended to support developers during the conversion of existing data structures to an object-oriented class design. This short description should show this tool and its possibilities to you. For context, you may want to refer to the article "From Relational To Object-Oriented Data Structures" in this issue.

Generating the Jasmine class structures requires running ODQL instructions on the server. Normally, the instructions are stored in script files, and then implemented on the server via the command line interpretor: *CODQLIE*. Creating these scripts can be a rather troublesome task, especially for illustrating the structure of existing databases in Jasmine's class hierarchy. One usually uses so-called CASE tools for such tasks, which are normally not included with object-oriented database, including Jasmine 1.21. While the Jasmine Script Utility does not claim to be such a CASE tool, it can none-the-less be helpful when converting dBase files, particularly since it is free. A future version is planned to support SQL tables, as well as xBase. Here now a short introduction to the functionality of this tool.

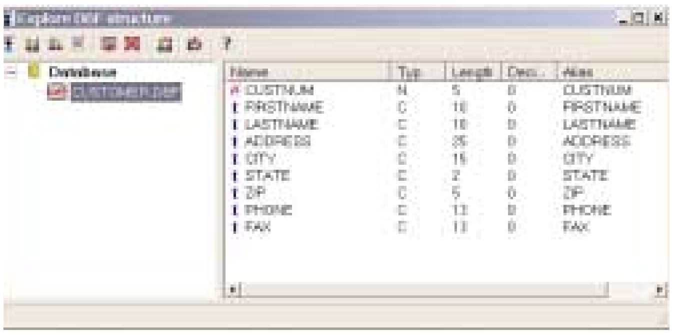
JSU start

When you have started the program, next to the help icon you find four toolbar icons at your disposal:

- starts the DBStructure Explorer.
- runs the Template Editor with a new template.
- opens the Template Editor with an existing Template.
- starts the ODQL Editor.

The DBStructure Explorer

Task of the DBStructure Explorers is it to read the structure of an existing file, to produce a template from that, and to store it. The intermediate step is a requirement, since there are very few cases, where the conversion is direct. Adjustments to the data type and field characteristics will be required much more often.

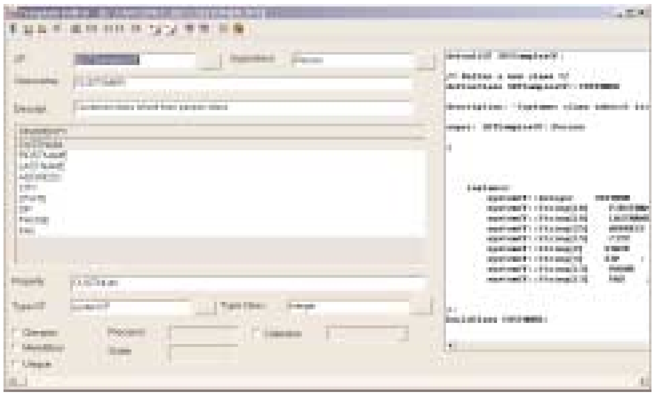


Name	Typ	Length	Deci...	Alias
CUSTNUM	N	5	0	CUSTNUM
FIRSTNAME	C	10	0	FIRSTNAME
LASTNAME	C	10	0	LASTNAME
ADDRESS	C	25	0	ADDRESS
CITY	C	15	0	CITY
STATE	C	2	0	STATE
ZIP	C	5	0	ZIP
PHONE	C	13	0	PHONE
FAX	C	13	0	FAX

With the Toolbar Icons , you can add and delete files from the Explorer. Selecting a file from the tree, and clicking on the icon, will create a template. All commands are also available from the context menu as well.

Template editor

The Template Editor is the real heart of the application. It can edit the generated templates, in order to create the necessary adjustments for the production of the ODQL script. Since most options are self-describing, I will talk about some few points here.



In order to be able to avoid spelling errors, with particular to capitalization, the Template editor can open a connection to a Jasmine Server and read the existing classes and properties from it. The icon builds this connection. Either clicking the icon or closing the Template Editor will close the connection.

The icons allow you to add new characteristics or delete them.
When the structure is ready, and all necessary fields are entered, the icon generates the ODQL script. Success is indicated by a checkmark in the right part of the window, and the script can then be stored via the icon.

Even though this utility prevents some errors from occurring, the user remains responsible for the correctness of the scripts. ODQL instructions are so powerful that it is nearly impossible to prevent all error combinations. Jasmine gives some consolation, in that it only accepts perfect scripts, and thereby guarantees the data consistency.

In Closing

On the enclosed CD, you will find both the executable application, with an install routine, and the complete source code of application. Please consider that programming style was not the focus this time around, rather, as in real life, the result. The source code gives you the ability to make modifications yourself. I am always interested in improvement suggestions.

Michael Zech
Email: mz@michaelzech.de

Traps, Tips n Tricks

Today we'll again try to answer some typical hotline questions.

Question:
I use DBFCDX as the default RDD. My tables have memo fields to store texts. Each time I call the DataWindow:Cancel() method existing texts in the memo field of the current record are lost. How can I work around this problem?

Response:
this problem is actually caused by a bug in the *DBServer:Refresh()* method. When called for the DBFCDX.RDD it can lead to loss of data in the memo fields. Although these memo fields contain regular text, they are interpreted as Blob. The problem can be easily worked around as shown below:

```
CLASS  DBSFix  INHERIT DBServer

METHOD Init (oFile,lShareMode,;
             lReadOnlyMode,xDriver,;
             aRdd) CLASS DBSFix
  LOCAL n         AS DWORD
  LOCAL x         AS USUAL

  SUPER:Init(oFile,;
             lShareMode,;
             lReadOnlyMode,;
             xDriver,;
             aRdd)
  FOR n := 1 TO SELF:wFieldCount
    IF aOriginalBuffer[2, n]
      x := SELF:FIELDGET(n)
      IF !IsNil(x)
        SELF:aOriginalBuffer[1,n] := x
      ENDIF
      SELF:aOriginalBuffer[2, n] := .F.
    ENDIF
  NEXT

  RETURN SELF
```

Question:
I use an extremely large amount of object instances in my application and at a certain point in time I receive the error message 'limit exceeded, Function RegisterAxit....' How large is the maximum number of registered Axit() methods and how can I change this number?

Response:
the maximum number of registered Axit() is set to 16000. This is the standard. Starting with the version 2.5 it can be adjusted in the following key in the Registry:

```
HKEY_CURRENT_USER
  Software
    ComputerAssociates
      CA-Visual Objects Applications
```

The numeric value named MaxRegisteredAxitMethods, if present, will determine the number of registered Axit() methods. To double the number of registered Axit() add this value to the above key and set it to 32000. To verify run the following test. Then change the value and run the same test again:

```
FUNC Start
    LOCAL n      AS INT
    LOCAL a      AS ARRAY
    LOCAL oTemp AS CTest

    a := {}

    DO WHILE .T.
        n++
        IF (n % 1000) = 0
            ? n
        ENDIF
        oTemp := CTest{}
        AAdd(a, oTemp)
    ENDDO

CLASS CTest

METHOD Init ()      CLASS CTest
    RegisterAxit(SELF)

METHOD Axit         CLASS CTest
    UnRegisterAxit(SELF)
```

Question :
I noticed that there are obviously two versions of the CDX driver, _DBFCDX.RDD and DBFCDX.RDD. What is the difference between both RDDs and what is the reason behind this concept?

Response:
the reason for this particular design is to achieve a high degree of flexibility. It uses the advantages of the RDD concept. The DBF file format differentiates three different file types:

Database files (DBF files)
Index files
Memo files

For each of these file formats there is a specific driver:

DBF, DBT:	cavodbf.rdd
CDX:	_dbfcdx.rdd
NTX:	dbfntx.rdd
FPT:	dbfcdx.rdd
DBV:	dbfmemo.rdd

Specifying the RDD (by name) determines the file formats to be used:

DBFNTX:	DBF, NTX, DBT
DBFMDX:	DBF, MDX, DBT
DBFCDX:	DBF, CDX, FPT

However at least 2 RDDs are actually always loaded. These can be varied. In addition the necessary RDD can be passed as an array instead of a name string. The following calls for the creation of a DBF file are absolutely equivalent:

```
DBCREATE("ntxdbt", a, "DBFNTX")
DBCREATE("ntxdbt", a, ;
          {"CAVODBF", "DBFNTX"})
```

Altogether the following combinations of file formats are possible:

```
FUNC Start
  FIELD nfield, mfield
  LOCAL a      AS ARRAY
  LOCAL i      AS INT

  a := { {“nfield”,”N”, 10, 0},{“mfield”,”M”, 10, 0} }
  SetAnsi(.F.)

  //  DBF, NTX, DBT
  DBCREATE(“ntxdbt”,  a, {“CAVODBF”, “DBFNTX”})

  //  DBF, NTX, DBV
  DBCREATE(“ntxflex”, a, {“CAVODBF”,”DBFMDX”,”DBFME-
MO”})

  //  DBF, MDX, DBT
  DBCREATE(“mdxdbt”,  a, {“CAVODBF”, “DBFMDX”})

  //  DBF, MDX, DBV
  DBCREATE(“mdxflex”, a, ;
        {“CAVODBF”, “DBFMDX”, “DBFMEMO”})

  //  DBF, CDX, FPT
  DBCREATE(“cdxfox”,  a, ;
        {“CAVODBF”, “_DBFCDX”,“DBFCDX”})

  //  DBF, CDX, DBV
  SET(_SET_MEMOEXT, “.DBV”)
  SET(_SET_MEMOBLOCKSIZE, 1)
  DBCREATE(“cdxflex”, a,;
          {“CAVODBF”, “_DBFCDX”, “DBFCDX”})

  RETURN
```

How can I determine the installed version of a LOE Automation Server

The following key in the registry holds the version of the automation interface:

HKEY_LOCAL_MACHINE\Software\Classes\Excel.Application\CurVer

The value found there is the version of the interface. For Excel97.Application for example 8 is returned. For other office applications the following keys can be used:

```
DEFINE REG_LMSC     :=
"HKEY_Local_Machine\Software\Classes\"
DEFINE REG_APPVER   := ".Application\CurVer"
DEFINE Reg_Excel    := (REG_LMSC + "Excel"+REG_APPVER)
DEFINE Reg_Word     := (REG_LMSC+ "WORD"+REG_APPVER)
DEFINE Reg_InternetExplorer := ;
  (REG_LMSC+"InternetExplorer"+ REG_APPVER)
DEFINE Reg_Powerpoint := (REG_LMSC+;
 "Powerpoint"+REG_APPVER)
DEFINE Reg_Outlook := (REG_LMSC+"Outlook"+REG_APPVER)
```

The method LV_Data reads the keys and return an array:

```
METHOD LV_Data() CLASS dlg_OfficeInfo
  LOCAL oRKey AS DcRegKey
  LOCAL cWert AS STRING
  LOCAL i AS DWORD
  LOCAL aTempdata AS ARRAY
aTempdata :=  {{"Excel", Reg_Excel},;
  {"Word", Reg_Word},;
  {"PowerPoint", Reg_Powerpoint},;
  {"Outlook",  Reg_Outlook},;
  {"Internet Explorer",; Reg_InternetExplorer}}
FOR i := 1 UPTO ALen(aTempdata)
  oRKey := dcRegKey{aTempdata[i,2]}
  oRKey:open()
  cWert := oRKey:defaultvalue
  oRKey:close()
  IF Empty(cWert)
    cWert := "n.v." // "n/a"
  ELSE
    cWert := ;
Right(AllTrim(cWert),1)
ENDIF
  aTempdata[i,2] := cWert
NEXT
SELF:aData := aTempdata
```

Adding Email to your Application

Erik Wynn

This session describes how to add automatic email capabilities to your application. SMTP will be presented, and we will discuss how to build an ActiveX server which communicates with SMTP servers. We will examine email message structures, multi-part messages, MIME types, base64 encoding, file attachments, and content types. A sound knowledge of CA-Visual Objects programming is useful for this session.

Email

mail1: m?l n. Middle English male from Old French from Old High German malhe.

2 a: something sent or carried in the postal system **b :** a conveyance that transports mail **c :** messages sent electronically to an individual (as through a computer system).

There is nothing new or mysterious about mail. Mail is a form of communication that has evolved over the past 8,000 years. The basic principle of communication is quite simple:

- a thought or idea originates inside a person's head
- the idea is encoded into a symbolic system and transcribed onto a suitable medium for transmission
- the idea is transmitted from the sender and is received by the receiver
- the encoded information is converted from the symbolic representation into its interpreted meaning

In fact, since prehistoric times, people have found increasingly more effective means of communicating over larger and larger distances. Signal fires on mountaintops announced awaited events. In Africa a sophisticated system of drum beating was used, where the tone and the rhythm determined the meaning.

Sometime around 8,000 years ago communication took a major turn for the better when ancient civilizations began making use of material to record written messages. These messages had the distinct advantage of being able to persist through time and space in a way that earlier forms of communication could not.

The rest, as they say, is history. Early courier systems for government and military use were organized in the Persian Empire under Cyrus, as well as in the Roman Empire and in medieval Europe. The arrival of the information age and the proliferation of personal computers made it possible to extend the simple concept of mail to its electronic form – email.

Email works on the same principle as that of prehistoric communication: an idea is encoded into a symbolic system, transcribed onto a suitable medium, transmitted over a wire, and received and decoded by the recipient. The rest of this paper deals with the minor details of how this actually takes place.

Why Automate?

Okay, so we might be able to add email to an application. But why? What advantage could this serve? Here are just a few examples to whet the appetite:

- Error handling and notification: when an error occurs in an application, the details of the error could be automatically sent to a system administrator or application developer
- Web site customer support: a customer could request information about a specific product or service, and automatically receive information in their inbox
- Automatic reminders: when was the last time you forgot someone's birthday, anniversary, or other important occasion?
- Asynchronous communication between two processes: any two processes can use email as an asynchronous means of remote inter-process communication.

Internet Protocol

There are many different kinds of email, ranging from small, proprietary systems to large corporate messaging solutions. By far, the most popular form of email is that involving the sending and receiving of electronic mail messages over the Internet. This paper deals with Internet-based email.

In order for two nodes on a network to communicate with one another, they must first decide on the language which they are going to use. This is called a protocol. There are a number of common protocols, covering a range of different networks and communication needs. Different protocols determine the way this communication takes place on the network.

In the Internet environment, the protocol is **T**ransmission **C**ontrol **P**rogram (TCP), which works on top of **I**nternet **P**rotocol (IP). This is referred to as TCP/IP. It should be noted that other protocols can also run on top of IP, such as **U**ser **D**atagram **P**rotocol (UDP), which is how DCOM works under Windows 95/98 and NT.

Services

Services are programs running using a certain protocol, which have non-proprietary standards. Common services determine how browsers view web sites (HTTP), how information is read from a newsgroup (NNTP), how files are uploaded and downloaded to an FTP site (FTP) how streaming multimedia content is delivered to a client (RSTP), and even how email messages are sent and received (SMTP and POP3).

Each of these protocols acts as a layer on top of a network protocol. For Internet-related services, the protocol is usually TCP/IP, however other protocols can be used.

Generally, each Internet service has been developed as a global effort, with one or more RFC (Request For Comment) publications proposing a standard for service implementation. The global nature of these proposed standards removes individual ownership issues, and helps to ensure longevity and consistency of implementation.

Winsock

Winsock, which stands for Windows Sockets, is a Microsoft Windows implementation of an IP stack, which enables windows clients to use any of the services across TCP/IP. The WSOCK32.DLL file contains the API functions available to clients using Winsock services. There are four groups of functions in the Winsock API:

- Conversion functions
- Database functions
- Socket functions
- Microsoft extensions

The Postman Delivers with SMTP

In order to add email automation to a program, one must first understand how email works. There are, in fact, a number of different ways to send email from within an application. MAPI (**M**ail **API**) is a client-side API which allows for the sending and receiving of messages. Using MAPI requires the MAPI.DLL and supporting DLLs to be installed on the client machine. For server machines, this may not be the case. A more specific solution, but one which is more readily available, is SMTP.

SMTP stands for **S**imple **M**ail **T**ransport **P**rotocol, and is the basis for sending email messages across the Internet. SMTP is a standard protocol for the exchange of electronic mail messages. It is based on RFC 821, published in August 1982, and has undergone only a few minor enhancements over the years.

SMTP can run across any streaming network protocol, including NITS, NCP, X.25, and TCP. TCP is by far the most common supported implementation today.

An SMTP Session

An SMTP session involves an SMTP transmitter and an SMTP receiver, both speaking SMTP. The sender initiates a conversation with the receiver, and an information exchange takes place. The SMTP commands define what can be said, how it can be said, and what the valid responses are. RFC 821 defines each of the SMTP commands in detail.

Sender

The sender is the originator of the message. The entire conversation between sender and receiver is decidedly one-sided: the sender issues a command and the receiver responds with a result code. The sender is initially some application program at the starting point of the message, such as Microsoft Outlook, Eudora, or some other email program. As the email message is transmitted through the Internet, a series of relays from one SMTP server to another takes place, in which case the receiver in one case becomes the sender is the next.

Reciever

The receiver is an SMTP server which receives the message from the sender. It is up to the receiver to implement a minimal set of SMTP commands. The sender can determine which commands the receiver supports before the conversation begins.

At any point during the conversation, the receiver and sender can trade places. This would be done between two SMTP servers relaying messages between different network regions, where the first server would pass on its message to the second, after which time the second server would pass on its messages to the first.

Minimal SMTP Commands

The following is a brief description of the basic SMTP commands that an SMTP server must support. A comprehensive description can be found in RFC 821 [1].

HELO

HELO or EHLO (see below) must be the first command sent from the transmitter to the receiver. This initiates a new SMTP session. During a session, zero or more email transactions can be conducted.

MAIL

Identify the mail sender. This does not need to be a valid email address! Many spammers have used this to their advantage. Many SMTP servers now require either the sender to have a local email account, or the direct recipient to have a local email account to prevent abuse of SMTP servers.

RCPT

Specify the mail recipient. This must be a local mailbox on the receiver SMTP server, or a mailbox on a remote SMTP server. If the SMTP server allows for the relaying of email messages, the SMTP receiver will attempt to relay the mail message to the destination SMTP server.

DATA

Provide the contents of the mail message. The mail data ends with a <CRLF>.<CRLF> sequence. It is inside the DATA section that the actual message header and body are defined. This is discussed later in this paper. See RFC 822 [2] for a detailed description of message format.

RSET (<u>R</u>e<u>SET</u>)

Abort the current email transaction. Other transactions can be conducted during this session.

NOOP

Do nothing, wait for OK response from receiver.

QUIT

Terminate SMTP session with receiver. Any pending email messages will be sent at this time.

Optional SMTP Commands

VRFY (<u>V</u>e<u>R</u>i<u>FY</u>)

Verify that a given email address exists on an SMTP server.

EXPN (<u>EXP</u>a<u>N</u>d)

Expand a mailing list. SMTP Servers maintain local user email account information, which can consist of mailing lists. The EXPN command will list all members of the mailing list specified.

SEND

Request that a mail message be sent directly to a user's terminal. The mail transaction is successful if the message is delivered the terminal.

SOML (Send Or MaiL)

Request that a mail message be sent directly to a user's terminal if the user is online, or to their mailbox if the user is not online. The mail transaction is successful if the message is delivered either to the terminal or the mailbox.

SAML (Send And MaiL)

Request that a mail message be sent directly to a user's terminal if the user is online, and, in all cases, to the user's mailbox. The mail transaction is successful if the message is delivered the mailbox.

TURN

A TURN command requests that the sender and receiver trade places: the sender becomes the receiver, and the receiver becomes the sender. This is useful for intermediate message relay stations for messages between different network regions.

A Sample Conversation with an SMTP Server

The following is a typical conversation between a client application, such as TELNET, and an SMTP server. In this session, the client sends a simple email message. Text entered by the client is presented in bold.

The <CRLF> character sequence, Chr(13) + Chr(10), is represented by the É character.

```
220 alpha.midtier.com NTMail
(v4.20.0009/NT2040.00.2c8043b9) ready for ESMTP
transfer

HELO www.midtier.com↵

250 alpha.midtier.com www.midtier.com

MAIL FROM:<someone@somewhere.com>↵

250 OK.

RCPT TO:<erik@midtier.com>↵

250 OK.

DATA↵
354 Start mail input; end with <CRLF>.<CRLF>.
To: Erik↵
From: Someone↵
Subject: Hi↵
↵
Hi There, ↵
↵
This is a quick message. ↵
↵
Bye↵
↵
.↵

250 OK.

QUIT↵

221 Goodbye www.midtier.com
```

Notice that the section following the DATA statement contains additional information, including a "Subject:" line, a "To:" line, a "From:" line, and a block of text which looks like a message body. The contents and layout of the DATA section is the next topic for discussion.

Message Format

Back in the early days of email, several informal standard mail message formats co-existed. This was back when Arpanet was the main network used (actually part of THE Internet) for the sending and receiving of email. A standard for a common message format was proposed in order to allow for a the exchange of email on a more global level. This standard was first published in RFC 733, and later in RFC 822. It describes the content and structure of the body of an email message (the section which appears after the SMTP "**DATA**" command).

The best way to describe the message format standard is to provide a brief sample. In fact, one sample has already been given, above. The following is a slightly more complex message.

```
Subject: Arpanet message format standard↵
From: "Erik Wynn" <erik@midtier.com>↵
To: "Someone OutThere" <someone@somewhere.com>↵
CC: "SomeoneElse Outthere" <someoneelse@somew-
hereelse.com>↵
Reply-To: "Erik Wynn" <erik@midtier.com>↵
Date: Mon, 23 Aug 99 14:20 EST↵
↵
Hi There, ↵
↵
This is a message to someone out there. ↵
↵
Is there anybody out there? ↵
↵
.↵
```

This message includes details for "cc" (carbon copy), and "Reply-To" email address, name information associated with the sender and receiver, and the date the message was sent. Notice the common pattern for the way each of these attributes is described.

Each email message consists of envelope information, such as Subject, From, To, etc…, as well as the actual message itself. A pair of CRLFs indicate the end of the envelope and the beginning of the email message.

There are a number of standard fields which can be used within the message envelope. The format is:

Field Name> ":" <Value>

The specific format of **<Value>** will be different depending upon what field is being described. For example, a sender or recipient specified in the "To", "From", "CC", "BCC" or "Reply-To" field has the format:

```
<Field Name> ":" "Name" <someone@somewhere.com>
```

Such as:

To: "Erik" <erik@midtier.com>

For multiple recipients, a separate entry for each recipient is required.

RFC 822 contains detailed information on the syntax for field descriptions.

A MIME is a Terrible Thing to Waste

MIME stands for **M**ultipurpose **I**nternet **M**ail **E**xtensions, and is a standard proposed in RFC 1521. It describes additional message header and body format extensions to allow the transmission of multi-purpose email messages. Among other things, MIME allows email attachments, different character sets, message encoding, and HTML formatted email messages.

Content-Type

The content-type determines what the body of the message contains. Sample content-types are text/plain, image/jpeg, audio/wav, and video/mpeg.
The content-type is specified in the message header as follows:

```
Content-Type: Text/plain
```

Character Set

The Character set is used to determine the way information in the message is encoded into a series of octets. Different languages typically use different character sets, so it is important to specify which character set was used to encode the message for it to be properly decoded. The character set is specified as a parameter of the content type, as follows:

```
Content-Type: Text/plain; charset=us-ascii
```

Or:

```
Content-Type: Text/plain; charset=iso-8859-6
```

Content-Transfer-Encoding

This field specifies how the message body will be encoded. Simple text messages can be sent without any kind of encoding, however other types of messages require encoding to convert their contents into a format which is suitable for transmission. Standard Content-Transfer-Encoding values are:

7bit
quoted-printable
base64
8bit
binary

And the content-transfer-encoding is specified as in:

Content-Transfer-Encoding: quoted-printable
A simple MIME message body is as follows:

```
From: Erik <erik@midtier.com>↵
To: Someone <someone@somewhere.com>↵
Subject: Test HTML Message↵
MIME-Version: 1.0↵
Content-Type: text/plain; charset=us-ascii↵
Content-Transfer-Encoding: 7bit↵
↵
Hi, ↵
↵
This is a regular email message.↵
Bye.↵
```

Simple Encoding

7Bit, 8Bit, and Binary encoding types all indicate that no encoding has been performed on the message body. However, there is a difference in the original contents between these types. 7bit encoding means that the data is all represented as short lines of US-ASCII data.. 8bit is the same as 7bit, however some of the characters may have the high-order bit set – creating some non-ASCII characters. Binary encoding means that high-order bits may be set, and that line lengths may not be short enough for SMTP transport (limited by some SMTP servers to be a maximum length of 76 characters).

Quoted-Printable Encoding

This is generally used to represent data with human-readable contents. Quoted-printable encoding converts the data so that it will not be modified by mail transport. The rules for implementing quoted-printable encoding are provided in RFC 1521 in detail.

Base64 Encoding

Base64 encoding is an popular method of encoding data that is not necessarily human-readable. This includes binary files, such as images, sounds, DLLs, etc…, or any file in proprietary format which is not necessarily human readable. The reason for encoding such data is so that the data can be represented without the use of high-order ASCII bits or syntactically significant control characters, such as CRLF, etc… which may prevent the mail from being transmitted properly. The basic process is as follows:

Data is broken up into 24 bit chunks (three 8-bit characters). These three characters are converted to binary, concatenated, and reinterpreted as four 6-bit chunks. A 6-bit chunk can hold numbers between 000000 and 111111 binary, or between 0 – 63 decimal. The mapped numbers are used as an index into a map to determine the mapped character. The following table determines the mapping:

0	**A**	**17**	**R**	**34**	**i**	**51**	**z**
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	48	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

Table 1: Base64 Character Mapping

A detailed description of the base64 encoding algorithm is provided in RFC 1521.

Multi-part messages

MIME also enables messages to be sent with multiple sections. In this case, each body part is placed within its own MIME section, and the MIME header information described above directly precedes the body part to which it applies. This means that different body parts can have different content-types, character sets, and encoding methods.

A typical application of this is to include message attachments as separate MIME parts. Multi-part messages are specified using:

```
Content-Type: multipart/mixed; boundary="—-MIME
BOUNDARY——-"
```

Where the "boundary" parameter is used to define a place marker or tag which will appear in the message body to separate different MIME sections.

Note that the phrase "This is a multi-part message in MIME format" appears in the message body before the first MIME section.

```
From: Erik <erik@midtier.com>↵
To: Someone <someone@somewhere.com>↵
Subject: Test HTML Message↵
MIME-Version: 1.0↵
Content-Type: multipart/mixed↵
       boundary="—-MIME BOUNDARY——-"↵
↵
This is a multi-part message in MIME format.
—-MIME BOUNDARY——-
Content-Type: text/plain; charset=us-ascii↵
Content-Transfer-Encoding: 7bit↵
↵
Hi, ↵
↵
This is a regular email message.↵
Bye.↵
—-MIME BOUNDARY——-↵
```

Message Attachments

Message attachments can be created by simply specifying additional MIME sections. The following MIME message shows how a message attachment can be defined:

```
From: Erik <erik@midtier.com>↵
To:  Someone <someone@somewhere.com>↵
Subject: Test HTML Message↵
MIME-Version: 1.0↵
Content-Type: multipart/mixed↵
       boundary="—-MIME BOUNDARY——-"↵
↵
This is a multi-part message in MIME format.
—-MIME BOUNDARY——-
Content-Type: text/plain; charset=us-ascii↵
Content-Transfer-Encoding: 7bit↵
↵
Hi, ↵
↵
This is a regular email message.↵
Bye.↵
—-MIME BOUNDARY——-↵
Content-Type: image/jpeg;↵
  name="VOCA.JPG"
Content-Transfer-Encoding: base64↵
Content-Disposition: attachment;↵
  filename="VOCA.JPG"↵↵
JJaFkL09mlJ65BbsPPuy09PtqjNa9+h/48vJhZLkem84FdpjUim
nNRzbv75
kL09mlJ65BbsPPuy09PtqjNa9+h/48vJhZLkem84FdpjUimnNRz
bv798jHs
65BbsPPuy09PtqjNa9+h/48vJhZLkem84FdpjUimnNRzbv798jH
========
↵
—-MIME BOUNDARY——-↵
```

Note: The base64 data in this sample is not intended to represent any actual image.

In this way, multiple attachments can be sent, each with their own content-type.

HTML Formatted Messages

Messages with different formatting syntax, such as RTF or HTML can also be sent using multipart messages. This is done using the multipart/alternative content-type. The basic idea is to create a message with multiple MIME sections, with each section containing the same data, but in different formats. The client agent (reader) will display the MIME section for the "highest" content-types supported, and will ignore those that it does not support. This allows each reader to render the message as well as possible for the reader. Here is an example:

```
From: Erik <erik@midtier.com>↵
To: Someone <someone@somewhere.com>↵
Subject: Test HTML Message↵
MIME-Version: 1.0↵
Content-Type: multipart/alternative↵
       boundary="—-MIME BOUNDARY——-"↵
↵
This is a multi-part message in MIME format.
—-MIME BOUNDARY——-
Content-Type: text/plain; charset=us-ascii↵
Content-Transfer-Encoding: 7bit↵
↵
<Normal text message>
É
—-MIME BOUNDARY——-
Content-Type: text/html; charset=us-ascii↵
Content-Transfer-Encoding: quoted-printable↵
↵
<Quoted-printable HTML message>
É
—-MIME BOUNDARY——-↵
```

The encoded HTML message is not shown in this sample.
This technique will ensure that HTML-capable email readers will render the HTML section of the message body, while non-HTML-capable email readers will use the plain text section.

Using Content-Type to Define Behavior

An interesting thing that can be done with email attachments is to sent the Content-Type value to the normal windows MIME type for that file type. This is useful since many email reader programs are able to use this MIME-type to determine which application to launch in order to view the attachment. For example, consider the following:

```
—-MIME BOUNDARY——-↵
Content-Type: image/jpeg;↵
  name="VOCA.JPG"
Content-Transfer-Encoding: base64↵
Content-Disposition: attachment;↵
  filename="VOCA.JPG"↵↵
JJaFkL09mlJ65BbsPPuy09PtqjNa9+h/48vJhZLkem84FdpjUim
nNRzbv75
kL09mlJ65BbsPPuy09PtqjNa9+h/48vJhZLkem84FdpjUimnNRz
bv798jHs
65BbsPPuy09PtqjNa9+h/48vJhZLkem84FdpjUimnNRzbv798jH
========
↵
—-MIME BOUNDARY——-↵
```

This tells the email reader that the MIME section contains an attachment of content-type image/jpeg. The email reader can then look this information up in the Windows Registry to determine which program should be launched to view this attachment.

It is easy to determine the MIME content type of a file attachment based on the file extension. The following VO program takes a file extension as a parameter and returns the associated MIME type.

```
FUNCTION GetMimeContentType( strFileExtension AS
STRING ) AS STRING PASCAL
    LOCAL liError      := 0         AS LONG
    LOCAL strMIMEType := “”        AS STRING
    LOCAL pszValue     := NULL_PSZ AS PSZ
    LOCAL dwKeyHandle := 0         AS DWORD
    LOCAL dwValueType := 0         AS DWORD
    LOCAL dwValueLen  := 0         AS DWORD

  liError := RegOpenKeyEx( HKEY_LOCAL_MACHINE,;
 String2Psz( “SOFTWARE\Classes\” + strFileExtension),;
      0,              ;  //  ulOPtions AS DWORD
      KEY_QUERY_VALUE,;  //  samDesired AS DWORD
      @dwKeyHandle    ;  //  phkResult AS PTR
      )//AS LONG PASCAL:ADVAPI32.RegOpenKeyExA#149

    IF ( liError == ERROR_SUCCESS )
       pszValue   := StringAlloc( Space( 2048 ) )
       dwValueLen := PszLen( pszValue )
       liError := RegQueryValueEx( dwKeyHandle,;
       String2Psz( “Content Type” ),  ;
                 0,;  //  lpReserved AS  DWORD PTR
       @dwValueType,;  //  lpType AS DWORD PTR
       pszValue,   ;  //  lpData AS BYTE PTR
       @dwValueLen ;  //  lpcbData AS DWORD PTR
    9 //  AS LONG PASCAL:ADVAPI32.RegQueryValueExA#157

       RegCloseKey( dwKeyHandle )
       IF ( dwValueType == REG_SZ )
           strMIMEType := Psz2String( pszValue )
       ENDIF
       MemFree( pszValue )
    ENDIF
 RETURN strMIMEType
```

A similar routine could figure out the registered application for the given MIME type.

Creating an Email Program

Now that we know all about Winsock, SMTP, email headers, and MIME message formats, we are equipped to be able to create our own email program. In this section, we will examine the steps required to automatically send email
There are three main issues an email program must deal with. Winsock issues, the SMTP conversation, and the MIME message format. We will examine each of these areas in more detail.

Winsock

At the system level, space must be allocated for windows sokkets and a winsock workarea, and a winsock stack must be initialized. A connection to the mail server must also be made. All this must take place before any kind of conversation can occur. Before we get to far into this, let me just say that VO 2.0 has a bug with the STRUCT definition for the _WINsockaddr_in structure. The correct structure definition is provided below:

```
STRUCT _WINsockaddr_in
  MEMBER  sin_family AS SHORT
  MEMBER  sin_port AS WORD
  MEMBER  sin_addr AS DWORD
  MEMBER  DIM sin_zero[8] AS BYTE
```

The following function shows the initialization and connection sequence, as well as the shut-down. This program uses blockin winsock calls. Non-blocking calls require an event loop to check return values.

```
FUNCTION SendMailMessage( strDomain AS STRING, ;
                 strFrom AS STRING, ;
                 strTo AS STRING, ;
                 strSubject AS STRING,
                 strMessage AS STRING ) ;
                 AS LOGIC PASCAL

  LOCAL pWSAData          AS _winWSADATA
  LOCAL pHostPtr          AS _winHOSTENT
  LOCAL pServPtr          AS _winservent
  LOCAL pSockPtr          AS _WINsockaddr_in
  LOCAL pAddrPtr          AS DWORD PTR

  LOCAL dwSockID          AS DWORD

  // Allocate storage FOR the Windows Sockets work
area.  Also allocate
  // storage for the socket.
  pWSAData := MemAlloc( _sizeof( _WINwsadata ) )
  pSockPtr := MemAlloc( _sizeof( _WINsockaddr_in ) )

  // initialize winsock
  IF WSAStartup( MAKEWORD( 1, 1 ), pWSAData ) == 0

    IF ( pHostPtr := gethostbyname( PSZ( strDomain ) )
) == NULL_PTR
      RETURN FALSE
    ENDIF

    IF(pServPtr := getservbyname(String2Psz(“smtp”), ;
      String2Psz( “tcp” ) ) ) == NULL_PTR
      RETURN FALSE
    ENDIF

    IF(dwSockID:= socket(AF_INET,SOCK_STREAM, 0 ) ) ;
      == INVALID_SOCKET
      RETURN FALSE
    ENDIF

    pAddrPtr := pHostPtr.h_addr_list

    pSockPtr.sin_port    := pServPtr.s_port
    pSockPtr.sin_addr    := DWORD( PTR( DWORD( pAddrPtr
) ) )
    pSockPtr.sin_family := AF_INET

    IFconnect(dwSockID,pSockPtr,;
       _sizeof(_winSockAddr_In ))<> 0
      RETURN FALSE
    ENDIF

    // rest of program
    // release memory
    MemFree( pWSAData )
    MemFree( pSockPtr )

    RETURN TRUE
```

SMTP

Once the winsock connection to the SMTP server has been established, it is a fairly easy task to conduct the SMTP conversation. This becomes a task of state-management. As each step of the conversation (state) is completed, the conversation moves on to the next state. A quick review shows the following state sequence:

HELO
MAIL FROM
RCPT TO
DATA
QUIT

The program must be able to read data from the windows socket and send data to the windows socket. This is done with the recv() and WSockSend() functions, respectively. To get data from the windows socket into a buffer, we must read until we reach at least one CRLF character sequence.

```
  LOCAL sBuffer           AS STRING
  LOCAL iRetLength        AS INT
  LOCAL sLine             AS STRING
  LOCAL sInputStream      AS STRING
  LOCAL symState          AS SYMBOL
  LOCAL siSMTPReplyCode   AS SHORTINT
  LOCAL dwPos             AS DWORD

  LOCAL dwSockID          AS DWORD

  // declare other variables, initialize winsock, etc…
  // initialize conversation state and input stream
  symState := #START
  sInputStream := “”

  WHILE TRUE
    // read until we get at least one CRLF
    sBuffer := Buffer( 128 )
    WHILE ! Instr( CRLF, sBuffer )

      sBuffer := Buffer( 128 )
      IF !(iRetLength:=recv(dwSockID,sBuffer,128,0x0 ) ) > 0
        RETURN FALSE
      ENDIF

      sBuffer := SubStr( sBuffer, 1, iRetLength )
      sInputStream := sInputStream + sBuffer
    END

    // walk forward to last line in stream
    WHILE Instr( CRLF, sInputStream )
      // extract line from input stream
      dwPos := At( CRLF, sInputStream )

      sLine := SubStr3( sInputStream, 1, dwPos - 1  )
      // keep the remainder for the next line
      sInputStream := SubStr2( sInputStream, dwPos +
SLen( CRLF ) )
    END

    // get SMTP reply code
    siSMTPReplyCode := Val( SubStr3( sLine, 1, 3 ) )
    DO CASE
    CASE siSMTPReplyCode = 220
      // HELO expected – send HELO and move to next
state
      sBuffer := “HELO “ + oMS:MailServer + CRLF
      IF WSockSend( dwSockID, sBuffer, SLen( sBuffer ),
0x0 ) <> ;
        SLen( sBuffer )
        RETURN FALSE
      ENDIF
      symState := #HELO

    CASE siSMTPReplyCode = 250

      // 250 OK  – check state, process
      DO CASE
      CASE symState == #HELO

        // send “MAIL FROM:”
        sBuffer := “MAIL FROM:<” + strFrom + “>” + CRLF

        IF WSockSend( dwSockID, sBuffer, SLen( sBuffer
), 0x0 ) <> ;
          SLen( sBuffer )
          RETURN FALSE
        ENDIF

        // advance to the next state
        symState := #MAIL

      CASE symState == #MAIL

        // send “RCPT TO:”
        sBuffer := “RCPT TO:<” + strTo + “>” + CRLF

        IF WSockSend( dwSockID, sBuffer, SLen( sBuffer
), 0x0 ) <> ;
          SLen( sBuffer )
          RETURN FALSE
        ENDIF
          // advance to the next state
        symState := #RCPT

      CASE symState == #RCPT

        // send “DATA”
        sBuffer := “DATA” + CRLF

        IF WSockSend( dwSockID, sBuffer, SLen( sBuffer
), 0x0 ) <> ;
          SLen( sBuffer )
          RETURN FALSE
        ENDIF
        // advance to the next state
        symState := #DATA
        CASE symState == #DATA

        // send “QUIT”
        sBuffer := “QUIT” + CRLF
        IF WSockSend( dwSockID, sBuffer, SLen( sBuffer
), 0x0 ) <> ;
          SLen( sBuffer )
          RETURN FALSE
        ENDIF
        // advance to the next state
        symState := #QUIT
          EXIT
      OTHERWISE
        // unknown state
        EXIT

      ENDCASE

    CASE siSMTPReplyCode == 354
      // send the mail headers and data
    OTHERWISE

      // unknown or unhandled SMTP reply code
      RETURN FALSE

    ENDCASE
  END
```

MIME Message Format

At this point, we have set up and are using a winsock connection to the SMTP server, and have managed the SMTP conversation using a state machine. Now all that is left is to embed the email message in the proper format.

For the purpose of the sample program, we will send a message of content-type text/plain, using the standard US-ASCII character set. The message will be sent using 7Bit Content-Transfer-Encoding.

We need to construct the message headers (envelope information) as follows:

```
  LOCAL strHeader AS STRING

  strHeader := "To:<" +  strTo + ">" + CRLF
  strHeader += "From:<" + strFrom + ">" + CRLF
  strHeader += "Subject: " + strSubject + CRLF
  strHeader += "MIME-Version: 1.0" + CRLF
  strHeader += "Content-Type: text/plain;" + CRLF
  strHeader += "  charset=US-ASCII" + CRLF
  strHeader += "Content-Transfer-Encoding: 7bit" +
CRLF

  // initialize winsock, conduct SMTP conversation
  // .
  // .
  // .

  // send message body
  sBuffer := strHeader + CRLF + strMessage + CRLF +
"." + CRLF

  IF WSockSend( dwSockID, sBuffer, SLen( sBuffer ),
0x0 ) <> ;
    SLen( sBuffer )
    RETURN FALSE
  ENDIF
```

Now wasn't that simple!

With a bit more work, this same program can be extended to allow for multiple recipients, cc and bcc recipients, alternate content types and character sets, different encoding methods, and multi-part mail messages with attachments or alternate message body formats.

This will be left as an exercise for the reader.

Conclusion

Adding automatic email capabilities to an application enhances its ability to reach out to its users. Email can be used to deliver automatic event notification, event reminders, and automatic customer feedback messages. It can also be used as a vehicle for asynchronous communication between two remote programs.

SMTP is the *de facto* Internet mail standard. When used in conjunction with the MIME standard message format, a wide variety of email message needs can be met. The RFCs are extremely valuable sources of information to the Internet developer wishing to embark on this effort. With these in hand, creating a program which manages the Winwock layer, the SMTP conversation, and the MIME message format is fairly straightforward. This same approach can be used to implement POP3 and NNTP (newsgroup) client programs.

Automating the way programs communicate, not just with their users, but with each other, brings a new degree of power to your application.

Bibliography

RFC 821: Simple Mail Transfer Protocol, August 1982, Jonathan B. Postel, Information Sciences Institute, University of Southern California.
RFC 822: Standard For The Format Of Arpa Internet Text Messages, August 13, 1982, David H. Crocker, Dept. of Electrical Engineering, University of Delaware.
RFC 1521: MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, September 1993, N. Borenstein, Bellcore, and N. Freed, Innosoft.
RFC 1869: SMTP Service Extensions, November 1995, J. Klensin, WG Chair, MCI, N. Freed, Editor, Innosoft International, Inc., M. Rose, Dover Beach Consulting, Inc., E. Stefferud, Network Management Associates, Inc. and D. Crocker, Brandenburg Consulting.
RFC 1522: MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text, September 1993, K. Moore, University of Tennessee.

RFC 1939: Post Office Protocol - Version 3, May 1996, J. Myers, Carnegie Mellon and M. Rose, Dover Beach Consulting, Inc.
RFC 977: A Proposed Standard for the Stream-Based Transmission of News, February 1986, Brian Kantor (U.C. San Diego) and Phil Lapsley (U.C. Berkeley).
RFC 1036: Standard for Interchange of USENET Messages, December 1987, M. Horton, AT&T Bell Laboratories, and R. Adams, Center for Seismic Studies.

Biography

Erik Wynn is a project manager and software architect with over 13 years of experience. He is president of Wynn Systems Development Group, Ltd, and specializes in object oriented application development in a Windows environment. He has experience with a wide range of Internet development products and technologies, and is one of the authors of the ***Active Web Toolkit™****, an Internet development product for the ASP/IIS environment. He is considered to be a world-leading CA-Visual Objects expert, and has acted as project manager, team leader, and project advisor for several large CA-Visual Objects applications. His interests include Internet technology, object-oriented analysis, design, and programming, Windows programming, and the integration of productivity tools in the development environment. He lives in the Canadian countryside, just outside of Ottawa with his wife Paule, his son Connor, and his daughter Keven. He can be reached on the Internet at ewynn@cactus-com.com.*

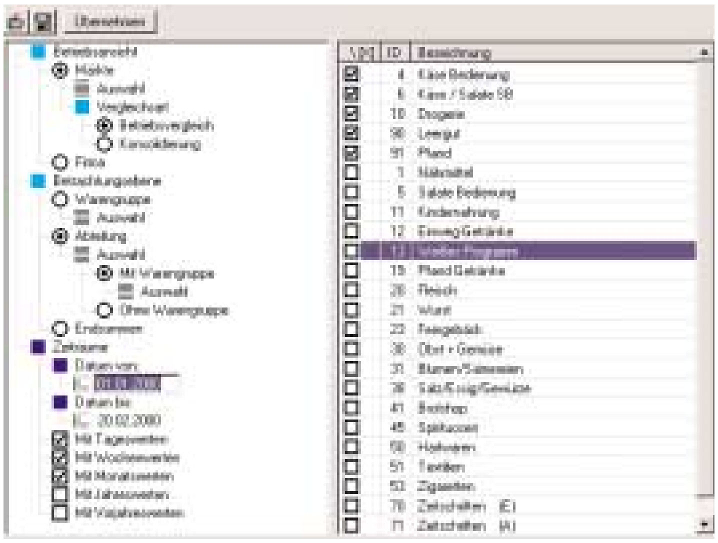
Ivo Wessel

CA-VO: Project “VO-SDT — the technic as quick as possible” (8)

After the article on SplitWindows, ListView and TreeView controls in the last issue, this article is dedicated to advanced techniques for programming of the two later-mentioned control clas-ses. These play an even larger role in modern applications, but they also must be correctly implemented, so that they demonstrate their complete efficiency and usefulness.

Overview

Apart from the event highlights in connection with cunning and TreeView controls, some typical programming techniques are demonstrated with these classes. Somewhat more unusual might be the generation of dynamic controls on TreeViews that permit the very flexible input of data in TreeViewItems. Finally the original label editing does not offer much clearance with ListView and TreeView controls too, there it is obvious to use your own control for this, that is placed at correct positions. This consequence settles the programming of the Wheel mouse apart from some practice tips — thus that flywheel, that finally halfway started also in VO 2.5a, and thus may no longer be missing in your applications under any circumstances.



Betriebsansicht
Märkte
Auswahl
Vergleichsart
Betriebsvergleich
Konsolidierung
Firma
Betrachtungsebene
Warengruppe
Auswahl
Abteilung
Auswahl
Mit Warengruppe
Auswahl
Ohne Warengruppe
Endsummen
Zeiträume
Datum von
01.01.2000
Datum bis
20.02.2000
☒ Mit Tageswerten
☒ Mit Wochenwerten
☒ Mit Monatswerten
☐ Mit Jahreswerten
☐ Mit Vorjahreswerten

[illegible]	ID	Bezeichnung
☒	4	Käse Bedienung
☒	6	Käse / Salate SB
☒	10	Drogerie
☒	90	Leergut
☒	91	Pfand
☐	1	Nährmittel
☐	5	Salate Bedienung
☐	11	Kindernahrung
☐	12	Einweg Getränke
☐	13	Weißes Programm
☐	15	Pfand Getränke
☐	20	Fleisch
☐	21	Wurst
☐	22	Feingebäck
☐	30	Obst + Gemüse
☐	31	Blumen/Sämereien
☐	38	Salz/Essig/Gewürze
☐	41	Bistshop
☐	45	Spirituosen
☐	50	Hartwaren
☐	51	Textilien
☐	53	Zigaretten
☐	70	Zeitschriften (E)
☐	71	Zeitschriften (A)

fig. 1: Dynamic SLE-Control on a TreeView

ListView controls

Event method ListViewItemChanged()

Frequently one would like to execute something additional to the (automatic) highlighting at the same time that the user selects a new line, thus a new ListViewItem. The method ListViewItemChanged() must therefore be imple-mented. In the following example, important interfaces are seen. oEvent:listViewItem is the selected ListViewItem object that should always be checked for “not a NULL_OBJECT”. It can be accessed on the ListView control via oEvent:control. This permits, for example, the poll in which ListView the event occurred, as everybody knows event methods have a window-central character and are called for all controls of the type concerned.

It is often overlooked with ListViewItemChanged() that it is called both when entering as well as when leaving a line item. The desired point in time (that will normally be oLVI:selected, thus the selection) is to be checked accordingly with the event object. One can easily check that actually two lines appear in the terminal window by an output in the IF poll—like “? Time(), oLVI:caption“. The last-mentioned nevertheless is obvious by the 25 “historical lines“ and is, by the way, already a part of the system library, thus it does not have to be linked additionally by merging the “Terminal Lite” library.

```
Method ListViewItemChanged (oEvent) Class winSplit
   Local oLVI As ListViewItem

   super:listViewItemChanged (oEvent)

   // oLVI := oEvent:control: ;
   //          getSelectedItem ()
   // Besser: LVI direkt vom
   // Ereignisobjekt holen
   oLVI := oEvent:listViewItem

   IF oLVI != NULL_OBJECT ;
      .AND. oLVI:selected ;
      .AND. oEvent:control == _oLivAutor
      self:FillListViewTitel (oLVI)
   ENDIF
```

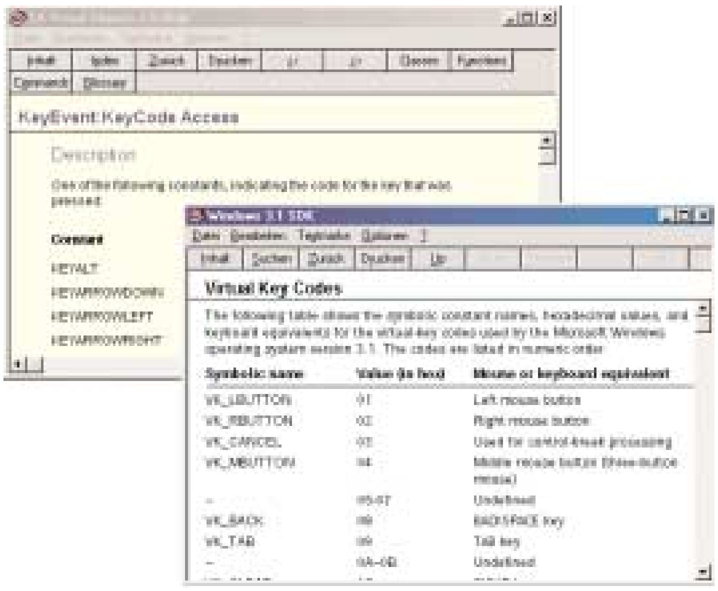
Event method ListViewKeyDown()

If one would like to react to pressed keys within a ListView control, this works at the simplest with the method ListViewKeyDown(). The following example shows the poll of the keys Tab and Shift+Tab. They provide with a ListView control as the “pane“ (thus area) of a Split Windows for the focusing of the next or previous Panes. With a Dialog- or a DataWindow, the focusing occurs automatically if the style (it is deactivated according to standard) is given to the ListView control in the Window wordprocessor “Tab Stop=TRUE”.

```
Method ListViewKeyDown (oEvent) CLass winSplit
   // Tastendruck im ListView-Control:
   // Fokussieren des nächsten bzw.
   // vorherigen Pane-Controls
   // mit Tab / Shift+Tab.
   super:listViewKeyDown (oEvent)

   DO CASE
   CASE oEvent:keyCode != VK_Tab
      // do nothing
   CASE GetKeyState (VK_Shift) < 0
      self:SetFocusToPreviousControl ()
   OTHERWISE
      self:SetFocusToNextControl ()
   ENDCASE
```

Who still possesses the old 16-Bit-aid of the Windows SDK — VO automatically installs the 32-bit version according to standard, in that unfortunately some useful tables are missing —, finds there some key codes, that the otherwise compatible KeyEvent description of the VO assistance does not mention.

More detail is of course provided in the MSDN Library whose acquisition (the cheapest basis version is usually completely enough) is strongly recommended.


KeyEvent:KeyCode Access

Description

One of the following constants, indicating the code for the key that was pressed.

Constant

KEYALT

KEYARROWDOWN

KEYARROWLEFT

KEYARROWRIGHT

Virtual Key Codes

The following table shows the symbolic constant names, hexadecimal values, and keyboard equivalents for the virtual-key codes used by the Microsoft Windows operating system version 3.1. The codes are listed in numeric order.

Symbolic name	Value (in hex)	Mouse or keyboard equivalent
VK_LBUTTON	01	Left mouse button
VK_RBUTTON	02	Right mouse button
VK_CANCEL	03	Used for control-break processing
VK_MBUTTON	04	Middle mouse button (three-button mouse)
–	05–07	Undefined
VK_BACK	08	BACKSPACE key
VK_TAB	09	TAB key
–	0A–0B	Undefined

fig. 2: KeyCode-Tables in the VO- and in the Windows-SDK-aid

The tiresome RETURN key...

Especially, the usually most important key, Return, does not release the event method ListViewKeyDown(). The key is to be polled within its own dispatcher of the control class; there a method can then be called with the owner of the control, thus the dialog, if necessary. The releasing controls are transferred to this key as a parameter for good reason.

Here this method is called OnReturnKey(), the delegation is constituent of the classes livStandard and trvStandard. Over the function IsMethod() it is naturally checked whether the owner of control has such a method at all. The following Return 1 “swallows“ all further messages, thus it is immediately branched back after executing the Owner method.

```
Method Dispatch (oEvent) Class livStandard
   // Bei einem Laschenfenster ohne Shell
   // kommt Return bei ListViewKeyDown ()
   // als einzige Taste nicht an!
   // Daher separat über Dispatcher mit
   // einem Aufruf einer OnReturnKey-
   // Callback-Methode.
   IF oEvent:message == WM_KeyUp
      DO CASE
      CASE oEvent:wParam == VK_Return ;
         .AND. IsMethod (self:owner,
                        (#OnReturnKey)
         Send (self:owner, ;
                        #OnReturnKey, self)
         Return 1
      ENDCASE
   ENDIF
   Return super:dispatch (oEvent)
```

Use of the context menu key

VO 2.5a also has learned something additional in the area of the keyboard operation in the meantime —however I’d still have a few ideas (and above all icons), but that is another topic... the context menu key is in the meantime fortunately polled and permits thus an efficient operation in the Repository Explorer. By the way, with F2 – like in the file Explorer – module names (—unfortunately they are still limited to 32 characters) can be renamed, a characteristic I estimate a lot. With Alt+Return you arrive rapidly at the Properties dialog.

```
   ...

   CASE oEvent:wParam == VK_Apps .OR. ;
      ( oEvent:wParam == VK_F1 ;
      .AND. GetKeyState (VK_Shift) < 0 )
      IF self:contextMenu != NULL_OBJECT
         self:contextMenu:showAsPopup ( ;
            self, self:origin)
         Return 1
   ENDIF
ENDCASE

   ...
```

If a ListView control has a context menu, it should naturally be possible to call it also over the keyboard. For those who still cling to old keyboards they have grown fond of, there should be an alternative to the menu key; there I prefer Shift+F1. Fans of a context-sensitive help can change over to Ctl+F1, or also Shift+F10.

Marking & focusing

A certain line of a ListView is marked by program control by setting the appropriate flag over the interface :selected with the ListViewItem. Usually also the focusing of this line (oLVI:focused) and the ListView control itself (over oLivControl:setFocus()) is recommended. In the following the flags must be updated with the ListView control, the substantially more rapid versions oLivControl:UpdateFocused (oLVI) etc., in comparison with oLivControl:setItemAttributes (oLVI) I already presented in the last issue, you find it in the module “_livStandard“ of the CA-VO2. By the way, you can also order old issues of the SDT if yours are missing...

The selection of a line takes place over oLivControl:getItemAttributes (nItem). It supplies with the ListViewItem object or NULL_OBJECT, the last-mentioned must always be polled here for safety’s sake. For nItem = 1, one receives the first line, whereas the last line would be nItem = oLivControl:itemCount. The class ListView has a method :EnsureVisible (nItem, lTeilweiseSichtbar) so that the marked line is also visible in each case, that provides for lTeilweiseSichtbar=FALSE for the line being is completely visible. If one already has the ListViewItem object, nItem can be transferred by means of oLVI:itemIndex.

Marking methods

A listbox whose ExStyle LVS_Ex_CheckBoxes ensures that StateImages appear in the form of check boxes that the user can activate over mouse-click, and a dummy key is suitable particularly well for the selection of items. In VO 2.5 this style can be switched on directly in the Window word processor on the “ExStyles” flat link. VO 2.0-user still must define it manually:

```
Define LVS_Ex_CheckBoxes := 0x00000004
```

In the CA VO21VO.AEF you find the appropriate lines in the module “_livStandard“. For marking individual or several lines it is best to write a central method that is called parameterized. This makes the repeated writing of same lines unnecessary, as for instance local control variables etc..

```
Method ListViewMarkCurrent () ;
   Class winSplit
   self:ListViewMark (#Current)

Method ListViewMarkAll () Class winSplit
   self:ListViewMark (#All)

Method ListViewInvertMark () Class winSplit
   self:ListViewMark (#Invert)

Method ListViewMark (symMode) Class winSplit
   // Universelle Methode zum Markieren
   // in ListViews.
   Local i, nLen  As DWord
   Local oControl As Object
   Local oLVI     As ListViewItem

   oControl := GetObjectByHandle ( GetFocus ())

   IF .NOT. IsInstanceOf (oControl, #ListView)
      oControl := _oLivAutor
   ENDIF

   IF symMode == #Current
      oLVI := oControl:getSelectedItem ()
      IF oLVI != NULL_OBJECT
         oLVI:stateImageIndex := ;
            IIF (oLVI:stateImageIndex == 1, 2, 1)
         oControl:UpdateStateImage (oLVI)
      ENDIF
   ELSE
      nLen := oControl:itemCount
      // Verhindert Flackern im Control.
      LockWindowUpdate (oControl:handle ())
      FOR i:=1 UPTO nLen
         oLVI := oControl:getItemAttributes (i)
         DO CASE
         CASE oLVI == NULL_OBJECT
            // do nothing
         CASE symMode == #All
            oLVI:stateImageIndex := 2
            oControl:UpdateStateImage (oLVI)
         CASE symMode == #Invert
            oLVI:stateImageIndex := ;
               IIF ( ;
               oLVI:stateImageIndex == ;
               1, 2, 1)
            oControl:UpdateStateImage (oLVI)
         ENDCASE
      NEXT i
      LockWindowUpdate (NULL_PTR)

   ENDIF
```

This technique of universal methods presents itself very frequently, like for switching on or off controls, for marking and de-marking etc.. In the following example below, the method :HighLighCurrentDay() has a logical parameter, that equips the method :UnHighlightCurrentDay() with FALSE; the default allocation is TRUE. Except in the method :UnHighlightCurrentDay(), the parameter is naturally never used. The example makes, nevertheless, the quite complex calculation of the area that should be drawn winRect unnecessary, although, as one sees, the “highlighting” differs clearly from the “opposite”.

```
Method UnHighlightCurrentDay () ;
   Class ccCalendar
   // Spart die doppelte Berechnung der
   // Zellen-Koordinaten etc.
   self:HighlightCurrentDay (FALSE)

Method HighlightCurrentDay (lHighLight) ;
   Class ccCalendar
   // Markiert oder entmarkiert den
   // aktuellen Tag _dCurValue.
   Local winRect Is _winRect
   Local nLeft, nTop, nCol, nRow As DWord
   Local cText As String

   Default (@lHighLight, TRUE)

   nRow  := Integer ((Day (_dCurValue) + ;
            _DoW (_dCurValue - ;
            Day (_dCurValue) + 1) - 2) / 7) + 1
   nCol  := _DoW (_dCurValue)

   nLeft := _nLeft + (nCol - 1) * _nCellWidth + 2
   nTop  := _nTop  + (nRow - 1) * (_nCellHeight + ;
            _oSetup:HeightSpace) + 1

   SetRect  (@winRect, nLeft, nTop, ;
      nLeft + _nCellWidth, ;
      nTop  + _nCellHeight)

   IF lHighLight
      // Bereich füllen und Tag schön
      // hell ausgeben
      FillRect  (_hDC, @winRect, ;
         _oSetup:BrushFocused)

      // Text einen Hauch nach links
      // verschieben, damit die Pixel der
      // Normaldarstellung exakt passend
      // sind -> Ausgleich der “+ 2” bei
      // Zuweisung an nLeft weiter oben.
      ShiftRect (@winRect, -2, 0)

      SelectObject (_hDC, _oSetup:FontDays)
      SetTextColor (_hDC, ;
         RGB (255, 255, 255))
      SetBkMode (_hDC, TRANSPARENT)

      cText := Str2 (Day (_dCurValue), 2)
      DrawText (_hDC, Psz (_Cast, cText), ;
         SLen (cText), @winRect, ;
         DT_VCENTER + DT_SINGLELINE + DT_RIGHT)
   ELSE
      // Bereich zum Neuzeichnen markieren
      // -> Aktuellen Tag ent-markieren.
      InvalidateRect (self:handle (), ;
         @winRect, TRUE)
   ENDIF
```

Event method ListViewMouseButtonDown()

The fact that an actual sufficiently plausible method can be quite tricky is shown by ListViewMouseButtonDown(). If one would like to find out whether a mouse-click on the left bit-map, the StateImage, has been caused —thus for instance the Checkbox with oLivControl:checkBoxes = TRUE — it depends on the switch FullRowSelect whether the Access PointOnItemStateImage actually supplies the desired value. If necessary, it must be determined whether the mouse position is situated within the first column. The whole line is marked —a ListView style, that I myself require strongly with the file

Explorer (praise for VO 2.5, that possesses a set UP switch here) —, supplies oEvent:pointOnItemStateImage also, then with TRUE, if the click occured outside of the first column — that is so tricky that it does not switch the CHECK box under any circumstances.

Here of course you have the advantage that such "features" are compensatable by calls of appropriate Windows API functions.

```
Method ListViewMouseButtonDown (oEvent) ;
   Class winSplit
   Local oLVI As ListViewItem
   super:listViewMouseButtonDown (oEvent)

   DO CASE
   CASE oEvent:isLeftButton ;
      .AND. oEvent:pointOnItemStateImage
      // Liefert nur bei :FullRowSelect
      // == FALSE ein korrektes Ergebnis!
      // Sonst oEvent:position auf
      //  > ListView_GetColumnWidth ( ;
      // oEvent:control:handle (), 0)
      // prüfen.
      oLVI := oEvent:listViewItem
      IF oLVI != NULL_OBJECT
```

Event method ListViewColumnClick()

If this column is to be sorted ascending or descending with one click in the column headers of a ListView, I would like to visualize the sort order in the column heading concerned and beeing clicked on by announcement of "/" or "\" (the Explorer of Windows 2000 has finally also learned that, therefore I hope for occasional installation also with VO 2.x...); that prevents reliably the repeated "for safety's sake still click another time" which can always be observed.

The event object supplies in addition apart from the ListView control (over the interface oEvent:control), also the clicked column and its symbolic name, that my method :SetSortColumn() of the class livStandard requires. In the figure 1 the ListView lines are by the way sorted according to the marking Checkbox (according to oLVI:stateImageIndex).

```
Method ListViewColumnClick (oEvent) ;
   Class winSplit
   // Klick in eine ListView-Spalte sortiert
   // diese Spalte aufsteigend bzw.
   // absteigend.
   super:listViewColumnClick (oEvent)

   oEvent:control:SetSortColumn ( ;
      oEvent:listViewColumn:nameSym)
```

Label column adjusted to the right

The so-called label of a ListView, the first column, can be edited; you know this characteristic also with the Windows Explorer. However it is not possible to format this first column adjusted to the right. As a solution one can define, however, an invisible column — the second (and first visible) one can then receive any adjustment.

A column is invisible if it has the length 0 and the heading NULL_STRING. But please make sure that — for instance in the method:AutoSize() of the class livStandard – such columns are never changed in their width.

Measure in pixels

In this connection the possibility is to be called of how one can receive the width of the column in pixels. oLivControl:getColumn (1):width e.g. supplies as well known with the width of the first column in characters. The following line supplies with the measure in pixels; as API function ListView_GetColumnWidth() is naturally zero-based; for the first column thus 0 would have to be transferred.

```
ListView_GetColumnWidth ( ;
   oEvent:control:handle (), nColumn - 1)
```

Owners of the VO-SDK (that then also have the source texts of the class library) know such API functions naturally from the module LISTVIEW.PRG. Since the source code unfortunately does not contain any comments, one should consult the MSDN LIBRARY also here; it contains detailed descriptions of the CommonControls and naturally all API functions.

Edit the label column

The editing possibility of the label is activated by setting a switch in the Window word processor. At run-time then, a "slow doubleclick" on the label column is sufficient. More comfortable & compatible is the same thing with F2; but also the activation over a PushButton or a menu option should be possible. The editing process is started with oLivControl:EditItemLabel(); this method is then called in the menu /Pushbutton method or in ListViewKeyDown().

The back storage of the input however does not take place automatically. For this it is rather necessary to overwrite the event method ListViewItemEdit() for the dialog concerned that supplies — as usual – the point in time of the beginning and the end of the editing process. The event object oEvent contains the necessary information: oEvent:editEnding is to be tested for writing back on TRUE; oEvent:editText contains the text, oEvent:listViewItem the ListViewItem object, thus the line of the edited cell.

Besides the appropriate API functions are naturally also available. Thus one receives access to the complete Edit control for instance over ListView_GetEditControl(). oEvent:lParam contains the pointer on the complete _winLV_DispInfo structure — who likes the API functions too much, could thus also test the text by the following line:

```
Method ListViewItemEdit (oEvent) ;
   Class dlgListensort

   Local winLV_DispInfo Is _winLV_DispInfo
   Local cText As String

   super:listViewItemEdit (oEvent)

   DO CASE
   CASE oEvent:editEnding
      winLV_DispInfo := ;
         Ptr (_Cast , oEvent:lParam)

      IF PszLen ( ;
         winLV_DispInfo.item.pszText) > 0
   Local cText As String

   super:listViewItemEdit (oEvent)
   DO CASE
   CASE oEvent:editEnding
      winLV_DispInfo := ;
         Ptr (_Cast , oEvent:lParam)
      IF PszLen ( ;
         winLV_DispInfo.item.pszText) > 0
         cText := Psz2String ( ;
            winLV_DispInfo.item.pszText)
```

The following example shows some possibilities of how the label could be written back into a column named #Pos:

```
Method ListViewItemEdit (oEvent) ;
   Class dlgListensort
   // Problem: Wird beim Editieren des
   // Label-SingleLineEdits Esc gedrückt,
   // muß der alte Wert restauriert
   // werden. Drückt man während des
   // Editierens Return, ohne den Wert
   // geändert zu haben, wird die Eingabe
   // gelöscht. Also darf im Falle
   // “Return gedrückt” nur dann die
   // Eingabe gespeichert werden, wenn
   // der neue Wert des Label-SLE ein
   // anderer ist als der ursprüngliche.
   Local oLVI  As ListViewItem
   Local cText, cBuffer As String
   Local hSle  As Ptr
   Local nLen As DWord
   Local lChanged As Logic

   super:listViewItemEdit (oEvent)
   oLVI := oEvent:listViewItem

   DO CASE
   CASE oEvent:editBeginning
      _nOldValue := oLVI:getValue (#Pos)
      oCCpshSortEingeben:disable ()

   CASE oEvent:editEnding
      lChanged := TRUE
      hSle     := ListView_GetEditControl ;
                    (oDClivData:handle ())
      cText    := oEvent:editText
      IF hSle != NULL_PTR
         nLen    := GetWindowTextLength (hSle)
         cBuffer := Space (nLen)
         GetWindowText (hSle, Psz (_Cast, ;
            cBuffer), nLen + 1)
         // ? “Edit-Control: “, cBuffer
      ENDIF
      // ? “cText: “, cText
      DO CASE
      CASE Val (cBuffer) == _nOldValue
         lChanged := FALSE
      CASE Val (cText) == 0
         oLVI:setValue (  0, #Pos)
         oLVI:setText  (“ “, #Pos)
         oLVI:stateImageIndex := 1
      OTHERWISE
         oLVI:setValue (Val (cText), #Pos)
         oLVI:setText  (     cText , #Pos)
         oLVI:stateImageIndex := 2
      ENDCASE
      IF lChanged
         oDClivData:setItemAttributes (oLVI)
      ENDIF
      oCCpshSortEingeben:enable ()
   ENDCASE
```

SingleLineEdits for each ListView column

However, one would often like to edit all columns, not only the label column. For this, one could place SingleLineEdit controls dynamically directly on the ListView control. But certainly one would have to ensure that the controls are drawn properly also with horizontal scrolling. In the following example I want to demonstrate the version that produces, in each case, one SLE at run-time below the columns, that possesses the respectively fitting measure. The function ListView_GetColumnWidth() that is necessary for it and returns the measure in pixels is already known.

Only the ListView control is to be placed in the Painter; the SLE are produced dynamically. If you placed a framework that surrounds the ListView control including its (later) input fields on the window, make sure that the Groupbox is transparent. In addition the Painter you set the transparency switch on the ExStyles latch on TRUE. The following figure shows the dialog in the Painter (without the SLE control) in the background, in the foreground the window at run-time with the column input fields.

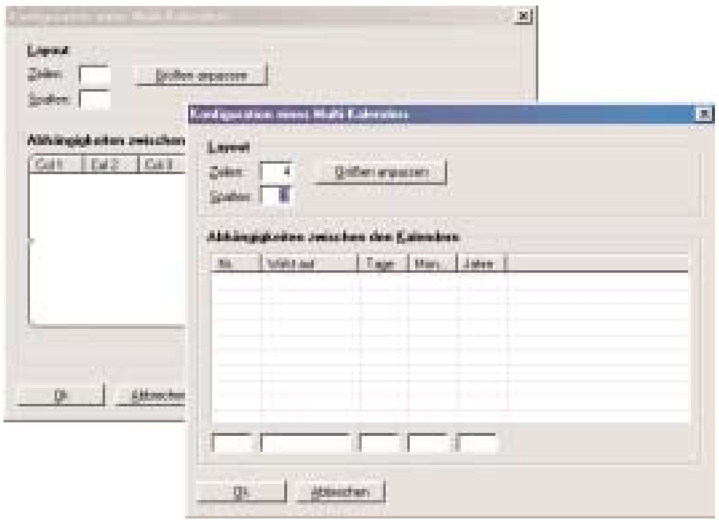


fig. 3: Dynamically generated SLE for each column

For the implementation we need an array as instance variable, in that we place the dynamically produced SLE control. When closing the dialog this array should be destroyed properly.

```
Class dlgCalendarConfig Inherit ;
      dlgCalendarConfig_vo
   Protect _aSle As Array

Method Init (oOwner) Class dlgCalendarConfig
   super:init (oOwner)
   _aSle := {}
   self:CreateListView ()
   self:CreateEditControls ()
   self:FillListView ()
   oDCsleZeilen:setFocus ()

Method Close (oEvent) ;
   Class dlgCalendarConfig
   super:close (oEvent)
   AEval (_aSle, { | oSle | oSle:destroy (), ;
      oSle := NULL_OBJECT })

   _aSle := NULL_ARRAY
```

```
Method CreateListView ()Class dlgCalendarConfig

   Local aColumn As Array
   Local i, nLen, nAlign As DWord

   aColumn := ;
      { { “Nr.”       , #Nr    ,  5 }, ;
        { “Wirkt auf”, #Wirkt , 10 }, ;
        { “Tage”     , #Tage  ,  5 }, ;
        { “Monate”   , #Monate,  5 }, ;
        { “Jahre”    , #Jahre ,  5 } ;
      }
   nLen := ALen (aColumn)

   FOR i:=1 UPTO nLen
      IF ALen (aColumn [i]) >= 4 .AND. ;
         IsNumeric (aColumn [i,4])
         nAlign := aColumn [i,4]
      ELSE
         nALign := LVCFmt_Left
      ENDIF

      oDClivAuswahl:addColumn ( ;
         ListViewColumn { ;
            aColumn [i,3], ;
            HyperLabel { ;
               aColumn [i,2], ;
               aColumn [i,1] }, ;
            nAlign })
   NEXT i
```

The columns of the ListView control are configured over an array that contains the column heading, the symbolic name and the measure (in characters). This configuration option I prefer a lot (frequent readers know that) permits a reading of the data from an external data base or INI file with only few modification.

```
Method CreateEditControls ()CLass dlgCalendarConfig
   // Damit die SLE unter dem Rahmencontrol
   // sichtbar sind, muß dieses im Painter
   // auf transparent gesetzt sein!
   Local i, nLen, nX, nY, nWidth As DWord
   Local oSle As SingleLineEdit
   Local oLVC As ListViewColumn

   nY    := oDClivAuswahl:origin:y - 28
   nLen  := oDClivAuswahl:columnCount
   nX    := oDClivAuswahl:origin:x
   _aSle := ArrayNew (nLen)

   FOR i:=1 UPTO nLen
      oLVC   := oDClivAuswahl: getColumn (i)
      nWidth := ListView_GetColumnWidth ( ;
                  oDClivAuswahl:handle (), i-1)
      oSle   := SingleLineEdit { ;
         self, -1, ;
         Point { nX, nY }, ;
         Dimension { nWidth - 5, 20 }, ;
         WS_Child + WS_TabStop + ;
         ES_AutoHScroll + WS_Border }
      oSle:hyperLabel := HyperLabel { ;
         oLVC:nameSym }
      oSle:picture := “####”
      oSle:show ()
      IF i == 1
         SetWindowPos (oSle:handle (), ;
            oDClivAuswahl:handle (), ;
            0, 0, 0, 0, ;
            SWP_NoMove + SWP_NoSize)
      ELSE
         SetWindowPos (oSle:handle (), ;
            _aSle [i-1]:handle (), ;
            0, 0, 0, 0, ;
            SWP_NoMove + SWP_NoSize)
      ENDIF
      _aSle [i] := oSle
      nX += nWidth
   NEXT i
```

The SingleLineEdit controls should be arranged horizontally below the ListView. The width of a SLE is the measure minus 5 pixels, so that they do not touch. The Picture masks of the input fields and all further characteristics could be quite stored in the column array, that would then however have to be defined as protect instance variable. In our simplified case, the SLE serves for the input maximum of four digits. Of course it would be considerable if necessary that not all ListView columns have to be also actually visible.

The input fields receive the necessary SLE styles; their name corresponds to that of the appropriate column. That will later facilitate data exchange between ListView column and input field enormously. We should however expect the fact that the sequence of the columns does not always& absolutely remain the one of initializing: The user could shift columns with the mouse. Our example will consider this case.

```
Method ListViewItemChanged (oEvent) ;
   Class dlgCalendarConfig

   Local i, nLen, nPos As DWord
   Local oLVI As ListViewItem
   Local symName As Symbol
   super:listViewItemChanged (oEvent)
   oLVI := oEvent:control: ;
           getSelectedItem ()

   IF oLVI != NULL_OBJECT ;
      .AND. oLVI:selected
      nLen := oDClivAuswahl:columnCount
      // Reihenfolge der SLE muß nicht
      // unbedingt mit der Reihenfolge
      // der Spalten übereinstimmen; wg.
      // Verschiebungsmöglichkeit etc.
      FOR i:=1 UPTO nLen
         symName := oDClivAuswahl: ;
            getColumn (i):nameSym
         nPos    := AScan (_aSle, ;
            { | oSle | oSle:nameSym == symName })
         IF nPos > 0
            _aSle [nPos]:value := Val ( ;
            oDClivAuswahl:getColumn (i): textValue)
         ENDIF
      NEXT i
   ENDIF
```

Consideration of the Tab sequence —also for dynamic controls

A problem (and naturally also its solution) is likewise in the production method. The input fields are to be arranged correctly — that means imperceptibly – in the Tab order. After producing the dialog however the two Pushbuttons “Ok” and “aborting” are naturally following in the Tab order after the ListView control. The function SetWindowPos() – again a API function already defined in Windows —ensures that the first SLE is sorted behind the ListView-, the others behind their predecessor SLE control. The Tab sequence is thereby ListView —first SLE..., last SLE, ok-Button.

Fill the SLE controls

Here once more meaningful & intelligent naming of the controls is necessary. Since the SingleLineEdits have the same names as the ListView columns, only the SLE control appropriate to the current column must be looked up and equipped in a loop by the columns. The AScan search in the SLE array costs only little more time, but they ensure the correct filling also for the case that the user shifted the columns. Also columns to that no SLE was assigned are so correctly treated. Make sure with programming that such boundary conditions are always fulfilled in favor of crash security.

Since the SLE —in our example —is to be updated when paging through the ListView control, we overwrite the event method ListViewItemChanged() that supplies us as we know with this point in time. Since the dialog receives only one ListView control (and also will not experience an armament in foreseeable time), I left out the poll on the aimed ListView control (in the type oEvent:name = = #livAuswahl) this time.

Now you will easily be able to implement in the “opposite direction”, the updating of the lines with a modification of a SLE control yourselves. The correct point in time would be the event method EditFocusChange (oEvent) with NOT. oEvent:gotFocus. Check whether there is a column with the name of the SLE control (oDClivAuswahl:getColumn (oEvent:nameSym != NULL_OBJECT). You receive the current line, an object of the class ListViewItem, by oLVI: = oDClivAuswahl:getSelectedItem(). If oLVI! = NULL_OBJECT, then the column can receive a new value by oLVI:setText (oEvent:control:textValue, oEvent:nameSym). Finally you must update the ListView control by oDClivAuswahl:UpdateText (oLVI).

Dynamic filling of ListView controls

The last consequence described the filling of ListView controls. Now the dynamic filling is to be described here with that not all lines (thus ListViewItems) are produced from the outset. In addition one defines a certain Buffer-quantity (about 100 lines), that is small enough, so that their reading in does not delay the structuring of the window noticeably. Anyway with a selection of for example 1000 lines (even if that is of course technically no problem) the GUI Design would have to be considered duly.

Further lines (let’s say, another 100 ListViewItems) can be loaded time-controlled in the background. Thus the user can already sight the volume of data, that is gradually completed or extended in peace. An alternative would be the attaching of a line “further...” as last ListViewItem with a hard disk symbol or something like that as an image to the data. With doubleclick on this ListViewItem — it is always easy to distinguish from “real” data by oLVI:itemIndex == oLivControl:itemCount or oLVI:imageIndex == IDI_Weitere etc.

Beside such “further...” – “Button“ of course also an “all” would be possible that reads in all data if the user knows what he is doing. A run beam with aborting possibility could make the time pass somewhat quicker for the user: As already said, you guarantee with the conception of such dialogs by suitable means that selection items are not “over-equipped”.

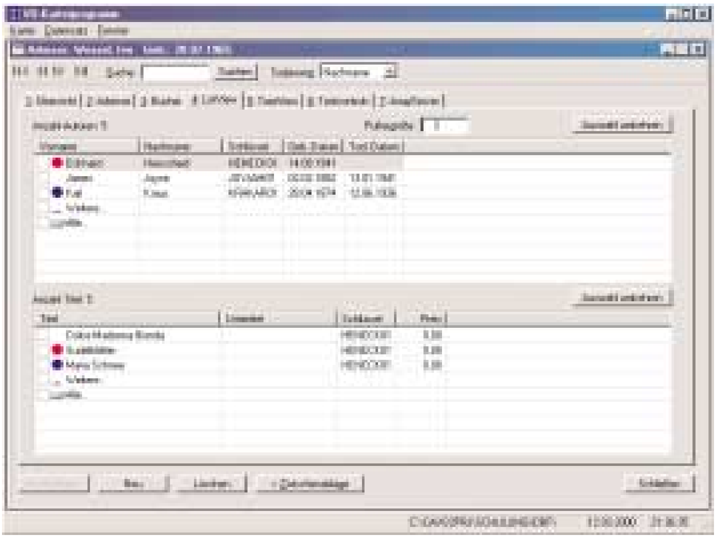


fig. 4: ListView with configurable Buffer-quantity and „reload“-Buttons

Alternative —virtual ListView...?

VO 2.5 and Windows make, in the meantime, as we all know a virtual ListView in the form of the DataListView class available that, similar to a Browser, also probably should replace this in the case of VO (this makes some necessary corrections at the Browser, as for instance the incorrect Scrollbar or display behavior unnecessary). Apart from some special features already addressed in the assistance virtual ListView controls have however not the very practical Checkboxes and are also not able to Muli Selekt. For these reasons I prefer the implementation of my own, let’s say, halfvirtual version. Ist actual-practical delimitations are, as mentioned, in the area of approximately four or five figures: But I don’t want to expect more selection items from the user anyway.

In the following example the ListView possesses an invisible column named #Recno in that ist sentence pointer position is stored per line. oLVI:getValue (#Recno) supplies thus with a number bigger than zero. The two last lines, “further...” and “all...” have a special bit-map in each case, but they also would be recognizable from their value:getValue (#Recno) = = 0.

The window possesses a server instance variable called _oDbServer. ListView controls can probably be also placed on DialogWindows —in the next consequence I will however show that one can place also a DataBrowser on DialogWindows. Doubleclick or Return on one of the two “reloading lines” causes a reading in of further records. In addition two methods ListViewMouseButtonDoubleClick (oEvent) and OnReturnKey() are to write — the last one, as is described in the front, released over the dispatcher of the livStandard class.

The method FillListView()

The number nMax of the maximal number of sentences to read can be specified over a SingleLineEdit also by the user (who has obviously no lack of knowledge). The method FillListView() is additionally called in the init method of the window, so that at least nMax records are visible from the outset. So that it is nice and universal and can be used directly for several ListView control on a window, it receives the ListView control, the server object and the sentence pointer number read in last.

If the third parameter is NIL, it is thus not transferred, this is for the method the signal that the call came from the init method and it can be started with reading in. Now available ListView entries are deleted.

Otherwise, if the server concerned has not yet reached oServer:eof, records are to be reloaded. Then at first the two last items, the "further..."- and the "all..."- Button from has to be deleted from the ListView. Please note that naturally first the last but one is to be deleted, then the lalt one. The alternative of deleting two times the last item is naturally just as correct, but it looks however very much like it was worth some optimization in the source code.

As the event method ListViewItemChanged() is internally released by deletion (or at all by changing) of ListViewItems, is to be guaranteed that this is only executed if the user —like by leafing through – has changed the line itself, e.g. "actively". The check on oLVI:focused carries that out; it is only TRUE if the line is focused.

Here again the outline of the actual filling method:

```
Method ListViewItemChanged (oEvent) ;
   Class tabKarteiListView
   Local oLVI As ListViewItem

   super:listViewItemChanged (oEvent)

   oLVI := oEvent:listViewItem
   // oEvent:listViewItem -> Aktuelle Zeile
   // oEvent:control -> Aktuelle ListView
   IF oLVI != NULL_OBJECT ;
      .AND. oLVI:focused
      .AND. IsNumeric (oLVI:getValue (#Recno)) ;
      .AND. oLVI:getValue (#Recno) > 0
      // „Echte“ Zeile mit Recno-Wert aktiv
      ...
      ENDIF
   ENDIF
```

Parameter of FillListView()

oLV ListView control (suitable thereby also for several LV)

oServer server objects (dto.)

nRecno next Recno or NILE to be read in

Structure of FillListView() for nRecno = = NIL

delete any ListView lines

oServer:goTop()

Structure of FillListView() for nRecno! = NIL

delete ListViewItem „further...“

delete ListViewItem „ all...“

oServer:recno: = nRecno

Reading routine of FillListView() for both versions

Read to nSaetze > = nMax or oServer:eof

If NOT. oServer:eof: attach Two lines „further ...“ and „all... “

If one clarifies this rather simple block structure, also the complete implementation of the method is not no longer complicated — if the limitation on even times 52 characters per print line with the SDT layout would not be there... The pictures of the reload Buttons are defined by two constants IDI_...; imagine that for instance a hard disk symbol is perhaps sensible for „further...“, and a "data base ton" or such a thing for "for all...".

if a server variable is used that possesses further Clients — other latch windows or such a thing — one should by of course ensure the data base run "in the dark" per oServer:suspendNotification() by renouncement of the notification of the other Clients. Surely then the protecting & restoring of the current sentence pointer would be sensible.

```
Method FillListView (oLV, oServer, ;
   nRecno) Class tabKarteiListView

   Local oLVI As ListViewItem
   Local i, nLen, nMax, nCount As DWord
   Local symCol As Symbol

   nMax   := Val (oDCslePuffer:textValue)
   nCount := 0
   nLen   := oLV:columnCount

   DO CASE
   CASE .NOT. IsNil (nRecno)
      // Die beiden "Weiter..." und
      // "Alle..."-Icons löschen
      // Löschen eines Items löst
      // ListViewItemChanged (oEvent) aus.
      oLV:deleteItem (oLV:itemCount - 1)
      oLV:deleteItem (oLV:itemCount)
      oServer:recno := nRecno
      oServer:skip ()
   CASE oLV:itemCount > 0
      oLV:deleteAll ()
      oServer:goTop ()
   ENDCASE
   IF oServer:eof
      oLVI := ListViewItem {}
      oLVI:setValue (0  , #Recno)
      oLVI:setText  (" ", #Recno)
      oLVI:setText  ("Keine Sätze vorhanden.", ;
         oLV:getColumn (1):nameSym)
      oLVI:stateImageIndex := 0
      oLV:addItem (oLVI)
   ELSE
      DO WHILE .NOT. oServer:eof ;
         .AND. nCount < nMax
         oLVI := ListViewItem {}
         oLVI:setValue (oServer:recno, #Recno)
         FOR i:=1 UPTO nLen
            symCol := oLV:getColumn (i):nameSym
            IF symCol != NULL_SYMBOL ;
               .AND. oServer:fieldPos (symCol) > 0
               oLVI:setValue (oServer:fieldGet ( ;
                  symCol), symCol)
            ENDIF
         NEXT i
         oLV:addItem (oLVI)
         nCount += 1
         oServer:skip (1)
      ENDDO
      IF .NOT. oServer:eof
         symCol := oLV:getColumn (1):nameSym
         oLVI := ListViewItem {}
         oLVI:setValue ( 0 , #Recno)
         oLVI:setText  (" ", #Recno)
         oLVI:setText  ("Weitere...", symCol)
         oLVI:imageIndex := IDI_Weiter
         oLV:addItem (oLVI)
         oLVI := ListViewItem {}
         oLVI:setValue ( 0 , #Recno)
         oLVI:setText  (" ", #Recno)
         oLVI:setText  ("Alle...", symCol)
         oLVI:imageIndex := IDI_Alle
         oLV:addItem (oLVI)
      ENDIF
   ENDIF
```

```
         oLVI:imageIndex := IDI_Weiter
         oLV:addItem (oLVI)

         oLVI := ListViewItem {}
         oLVI:setValue ( 0 , #Recno)
         oLVI:setText  (" ", #Recno)
         oLVI:setText  ("Alle...", symCol)
         oLVI:imageIndex := IDI_Alle
         oLV:addItem (oLVI)
      ENDIF
   ENDIF
```

Sorting of almost all lines

If such a ListView control equipped with two "final lines" in form of the reload Buttons is sorted by column-clicks, it is of course to be guaranteed that "further..." and "all..." maintain their position, thus they are not sorted with the others. Since VO permits the definition of any sort methods, that is very simply possible. The comfort of being able to write the sort routine in VO code costs some OOP overhead that leads to the fact that the assortment of about 1000 files in the Windows Explorer actually does not pose any problem, but the thing already becomes rather slowly-acting starting from 200 entries in a VO ListView – if the sorting routines are not implemented C-like.

The event method ListViewColumnClick(), like described above, causes the column assortment. By a poll of the column name also the use of your own sort method for each column would be possible. The class livStandard contains numerous examples.

```
Method ListViewColumnClick (oEvent) Class winSplit
   // Klick in eine ListView-Spalte
   // sortiert diese Spalte aufsteigend
   // bzw. absteigend.
   super:listViewColumnClick (oEvent)

   IF oEvent:listViewColumn:nameSym == #Betrag
      oEvent:control:SortingUpMethod   :=
#SortingUpVal
      oEvent:control:SortingDownMethod :=
#SortingDownVal
   ELSE
      oEvent:control:SortingUpMethod   := #SortingUp
      oEvent:control:SortingDownMethod := #SortingDown
   ENDIF
   oEvent:control:SetSortColumn ( ;
      oEvent:listViewColumn:nameSym)
```

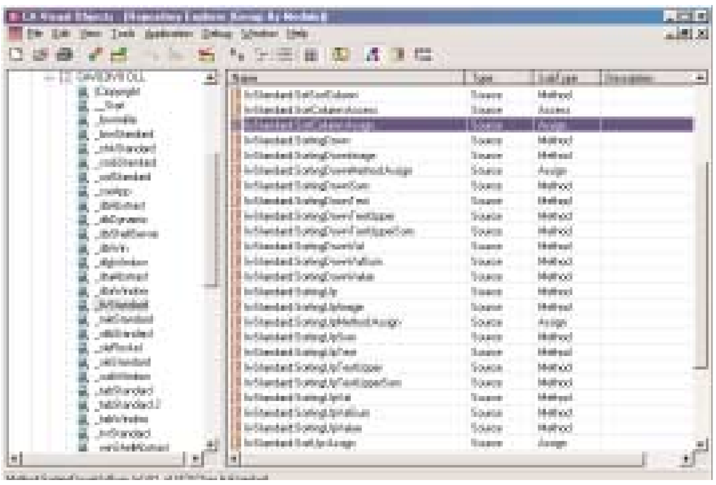


fig. 5: a view into the sortin-routines of the livStandard

For our case the condition of the distinction of that sort method is used would depend on the oServer:eof status. Also the check for oLivControl:getItemAttributes (oLivControl:itemCount): itemIndex == IDI_Alle leads to the same distinction. Two sorting routines that ignore the two last lines, but sort otherwise the remaining lines in a ascending or descending way, are then looking like this:

```
Method SortingUpVirtual (oLVI1, oLVI2) ;
   Class livStandard
   // Callback-Methode bei Aufruf von ;
   // :sortItems () (z.B. in der Methode
   // ListviewColumnClick ()); sorgt für
   // eine aufsteigende Sortierung der
   // Einträge, ignoriert die letzten
   // beiden Zeilen.
   Local u1, u2 As Usual
   Local nIgnore As DWord

   IF _symSortCol != NULL_SYMBOL
      u1 := oLVI1:getValue (_symSortCol)
      u2 := oLVI2:getValue (_symSortCol)
      nIgnore := self:itemCount
      DO CASE
      CASE u1 == u2 ;
         .OR. oLVI1:itemIndex == nCount   ;
         .OR. oLVI2:itemIndex == nCount   ;
         .OR. oLVI1:itemIndex == nCount-1 ;
         .OR. oLVI2:itemIndex == nCount-1
         Return  0
      CASE u1 <  u2
         Return -1
      OTHERWISE
         Return  1
      ENDCASE
   ELSE
      Return 0
   ENDIF

Method SortingDownVirtual (oLVI1, oLVI2) ;
   Class livStandard
   // Wie oben, aber absteigende Sortierung.
   Local u1, u2 As Usual
   Local nCount As DWord
   IF _symSortCol != NULL_SYMBOL
      u1 := oLVI1:getValue (_symSortCol)
      u2 := oLVI2:getValue (_symSortCol)
      nCount:= self:itemCount
      DO CASE
      CASE u1 == u2 ;
         .OR. oLVI1:itemIndex == nCount   ;
         .OR. oLVI2:itemIndex == nCount   ;
         .OR. oLVI1:itemIndex == nCount-1 ;
         .OR. oLVI2:itemIndex == nCount-1
         Return  0
      CASE u1 <  u2
         Return  1
      OTHERWISE
         Return -1
      ENDCASE
   ELSE
      Return 0
   ENDIF
```

A similar technique is also used if the ListView control contains for example a sum as the last line in each case, that should likewise not be sorted with the others: here it should be checked only for self:itemCount in the first CASE.

Time-controlled dynamic reloading of ListViewItems

Timers can be announced in VO 2.5 (which has not been documented before) also over the window method:RegisterTimer(). However only whole second values can be transferred here for inexplicable reasons (particularly in the age of the gigahertz processors...), that limits the resolution to 1 second. The resolution in Windows amounts to nevertheless about 50 msec, so that values of 100 or 200 msec can be quite useful also with data base accesses. The aid conceals by the way also that a timer can be reset also without closing of the window by RegisterTimers(). When destroying the window a possibly defined timer is however naturally terminated automatically.

The good, classical version over SetTimers(), KillTimer and poll of WM_Timer in the window dispatcher is naturally also continuing to be possible, offers the mentioned resolution of 50 msec and should therefore be used here. By the way such timer vents arrive always at the main window with use in a latch window (thus a window of a Tab-control); therefore the example of such a latch window, that is informed by the main window about the method OnTimer() of the end of the time interval should be implemented.

```
Method Init (oOwner, nCtrlID, oServer) ;
   Class dtaAutor
   super:init (oOwner, nCtrlID, oServer)

   SetTimer (self:handle (), 1, ;
      1000, NULL_PTR)

Method Close (oEvent) Class dtaAutor
   KillTimer (self:handle (), 1)
   super:close (oEvent)

Method Dispatch (oEvent) Class dtaAutor
   IF oEvent:message == WM_Timer ;
      .AND. IsMethod (oDCtabControl:currentPage, ;
         #OnTimer)
      Send (oDCtabControl:currentPage, ;
         #OnTimer)
   ENDIF

   Return super:dispatch (oEvent)
```

The class tabAutor_ListView has two instance variables _oDbAutor (an object of a server class like for example dbAutor) and _nNextRecno; the last DWord value is 0 at the beginning (this provides for the necessary „first deleting“ of possibly available data) and then contains in each case the Recno of the next data record to be read. Otherwise the methodology resembles naturally very much to the “halfvirtual“ version; per timer flow a next data record is added to the ListView here so long, until the sentence pointer reached EOF. As said; the combined application of a certain Buffer number would be surely useful here.

```
Method OnTimer () Class tabAutor_ListView
   // Siehe Dispatch () der Klasse dtaAutor
   // -> Wird alle 1000 msec aufgerufen.
   // self:owner:caption := Time ()
   IF .NOT. _oDbAutor:eof
      self:FillListViewWithTimer ( ;
         oDClivAutor, _oDbAutor)
   ENDIF

Method FillListViewWithTimer (oLV, ;
   oServer) Class tabAutor_ListView

   Local oLVI As ListViewItem
   Local i, nLen, nRecno As DWord
   Local symFeld As Symbol
   Local nFocus As Ptr

   nFocus := GetFocus ()
   nLen   := oLV:columnCount

   IF oLV:itemCount > 0 ;
      .AND. _nNextRecno == 0
      oLV:deleteAll ()
   ENDIF
   nRecno := oServer:recno
   oServer:suspendNotification ()
   IF _nNextRecno == 0
      oServer:goTop ()
   ELSE
      oServer:recno := _nNextRecno
   ENDIF
   IF oServer:eof
      // Jetzt könnte z.B. auch der Timer
      // des dtaAutor-Fensters
      // zurückgesetzt werden, weil
      // die Liste vollständig
      // eingelesen worden ist.
      // _nNextRecno := 0
   ELSE
      oLVI := ListViewItem {}
      FOR i:=1 UPTO nLen
         symFeld := oLV:getColumn (i):nameSym
         IF oServer:fieldPos (symFeld) > 0
            oLVI:setValue ( ;
               oServer:fieldGet ( ;
               symFeld), symFeld)
         ENDIF
      NEXT i
      // Recno merken in der unsichtbaren
      // Spalte (die keinen Text hat!)
      oLVI:setValue (oServer:recno, #RECNO)
      oLVI:setText  (“”           , #RECNO)
      oLV:addItem (oLVI)
      // 500 msec Nixtun simulieren, damit
      // man sieht, wie die Sätze
      // zeitverzögert eingelesen werden.
      // Sleep (500)
      // Sleep (100)
      oServer:skip (1)
      _nNextRecno := oServer:recno
   ENDIF
   oServer:recno := nRecno
   oServer:resetNotification ()
   // Alle Spaltenbreiten werden optimiert
   oLV:AutoSize ()
   IF nFocus != NULL_PTR
      SetFocus (nFocus)
   ENDIF
```

Since reading in the record should really not occur in the background and not influence the acting of the user in the window, the loop notes at the beginning the active control, that is finally focused again.

TreeView controls

Event method TreeViewSelectionChanged()

Like with ListView controls a “line change”, thus the moving within the control, can be intercepted over the event method. In the case of a TreeView the event object oEvent possesses two interfaces oEvent:oldTreeViewItem and oEvent:newTreeViewItem, over that the old (let's say, the abandoned) and the new selected TVI is in the access. The following example displays the active TVI with fat Font; when leaving this is to be reset again. Also here one should always test on "oTVI != NULL_OBJECT", since otherwise a poll like oTVI:bold causes a crash for boundary conditions that are perhaps rare, but in practice nevertheless occurring, if oTVI is NULL_OBJECT.

```
Method TreeViewSelectionChanged (oEvent) ;
   Class winSplit
   // Aktuelle Zeile in fettem Font
   Local oTVI As TreeViewItem

   super:treeViewSelectionChanged (oEvent)
   oTVI := oEvent:oldTreeViewItem
   IF oTVI != NULL_OBJECT
      oTVI:bold := FALSE
      oEvent:control:setItemAttributes (oTVI)
   ENDIF
   oTVI := oEvent:newTreeViewItem
   IF oTVI != NULL_OBJECT
      oTVI:bold := TRUE
      oEvent:control:setItemAttributes (oTVI)
   ENDIF
```

Expanding and collapsing

There is another useful version to expand a TreeView-Control that is not in the IDE: the possibility to collapse the inner nodes but to expand the root elements. In the VO Tree the project names would then be visible, their applications & libraries however invisible. With each creation or deletion of a project (that is supposed to occur sometimes times; and I also have more than one or two projects...) all nodes are expanded, after that I tend to restart VO again; that can be done much faster than the toilsome manual collapsing of the projects.

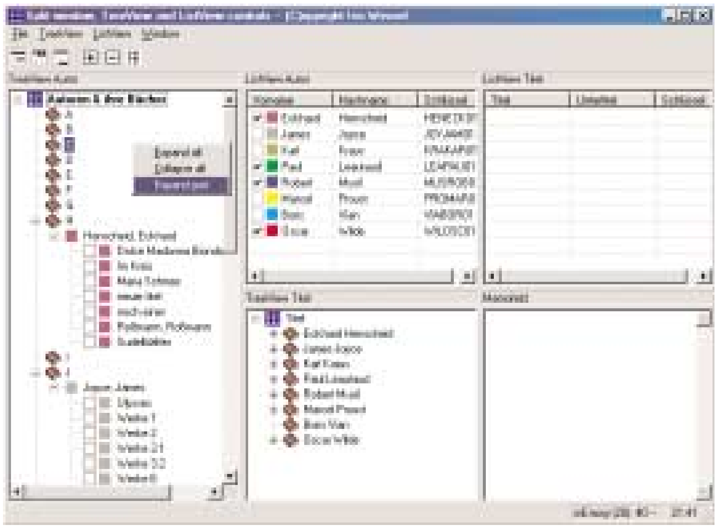


Fig. 6: Thein a moment only the letter elements A to Z will be seen...

If the TreeViewItem, that one inserts into the root is for example called #_Root, such a neat expanding of the oTreeView:expand (#_Root) is possible very easily. Here a few describing code lines:

```
oTVI := treeViewItem { #_Root, "Visual Objects 2.5" }
oTreeView:addItem (#Root, oTVI)
```

One could then also eliminate a second unpleasantness in the VO-IDE: During recursive collapsing of a tree one should ensure by LockWindowUpdate() that Windows does not draw the Tree again. This API function permits the locking of a window or a control; in the case of a TreeView-Control the items and above all the usually hyperactively bouncing scrollbar are updated as the last thing, after renewed call of the function with parameter NULL_PTR. My two methods from the CAVO2IVO call a common recursive method ExpandCollapseAll(), that use this function.

```
Method CollapseAll () Class trvStandard
   // Alle Ebenen zuklappen
   Local oTVI As TreeViewItem

   LockWindowUpdate (self:handle ())

   oTVI := self:getRootItem ()
    DO WHILE oTVI != NULL_OBJECT
       self:ExpandCollapseAll (FALSE, oTVI)

       // Nächstes Kind im Tree
       oTVI := oTVI:nextSibling
    ENDDO
   LockWindowUpdate (NULL_PTR)

Method ExpandAll () Class trvStandard
   // Alle Ebenen aufklappen
   Local oTVI As TreeViewItem

   LockWindowUpdate (self:handle ())

   oTVI := self:getRootItem ()
    DO WHILE oTVI != NULL_OBJECT
       self:ExpandCollapseAll (TRUE, oTVI)

       // Nächstes Kind im Tree
       oTVI := oTVI:nextSibling
    ENDDO
   LockWindowUpdate (NULL_PTR)
```

The common recursive function ExpandCollapseAll ()

The first parameter lExpanding controls the behavior of the method; TRUE expands the branch, FALSE collapses it accordingly. Since the expanding/collapsing of the nodes only differs by this value, a common method is once more very usful.

```
Method ExpandCollapseAll (lExpanding, ;
   oTVI) Class trvStandard
   // Alle oder einen Zwei zu- oder
   // aufklappen
   IF oTVI == NIL
      // Kein Parameter angegeben:
      // Ist ein TVI gewählt?
      oTVI := self:getSelectedItem ()

      DO CASE
      CASE oTVI == NULL_OBJECT
         // Nein -> Dann alle auf-/
         // zuklappen; ab der Wurzel.
         oTVI := self:getRootItem ()

      CASE oTVI:value != NIL
         // In ein Nicht-Knotenelement
         // geklickt
         // -> Suche Elternelement
         oTVI := self:getParentItem (oTVI:nameSym)
      ENDCASE
   ENDIF
   IF lExpanding
      oTVI:expand ()
   ELSE
      oTVI:collapse ()
   ENDIF
   oTVI:treeViewControl:setItemAttributes (oTVI)

   oTVI := oTVI:firstChild
   DO WHILE oTVI != NULL_OBJECT
         // Sich rekursiv durch den Baum
         // hangeln...
         self:ExpandCollapseAll ( ;
            lExpanding, oTVI)
         oTVI := oTVI:nextSibling
   ENDDO
```

Marking & focusing

Selecting such TreeViewItems, that are invisible as in collapsed branches of the tree, is almost amazingly simple and automatical. You only have to hand over the (unique!) name of the desired items as a parameter to the method oTreeView:selectItem (symName). The control then ensures for the fact that the nodes are expanded accordingly, and the item appears within the visible area.

Thus for instance a dialog is to be seen in the following figure, that sorts all controls of a window into a Tree by the push of a button, whereby for each available type of control such as Pushbutton, SingleLineEdit etc. a node is produced . If the user now marks a control on the main mask (it is bordered, that one should also be able to detect on the screenshot), it appears in the Tree control with its user rights data. With this, all controls can be provided with user right-specific data such as “display”, “hide”, “disable” etc. at runtime in each window. Beside the selection on the mask itself (naturally also multiselect, lassoing etc. is possible here) controls can also be specified over the Tree. From here thus all PushButtons canbe marked with one click.

If you ask yourself how to click on the controls without activating these (also the mouse pointer remains the normal arrow): No, the control classes are not derived individually (the dialog is to function on *each* window...): the controls will be provided with the style WS_Disabled on push of a button. They then remain nicely rich in contrast (in contrary to:disable()), and mouse clikks etc. now directly arrive at the window without detour over the control.

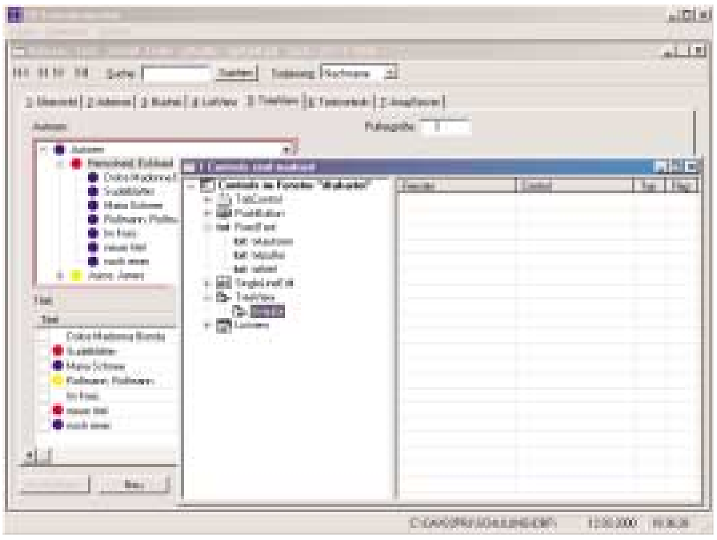


Fig. 7: oTreeView:selectItem() expands the Tree if necessary.

Dynamic filling of TreeView-Controls

With this version—normal filling was already described in the last edition—the reading of the node data only takes place if the node is expanded by the user. With a very large quantity of data (however as one can see for example at the registry editor, is actually hardly critical with Windows the 9x, NT, 2000 etc.) one could remove the data when collapsing again.

The central method here is the event method TreeViewItemExpanded(), that is called , if a node is expanded or collapsed. Over oEvent:collapsed or oEvent:expanded one can decide whether data is to be read. oEvent:treeViewItem contains the current line in the TreeViewItem object.

```
Method TreeViewItemExpanded (oEvent) ;
   Class winSplit
   // Icon “Buch” umschalten
   Local oTVI As TreeViewItem

   super:treeViewItemExpanded(oEvent)

   oTVI := oEvent:treeViewItem

   IF oTVI != NULL_OBJECT ;
      .AND. oTVI:nameSym != #_Root
      // Nur bei Klick auf Nicht-
      // Wurzel-Element
      IF oEvent:collapsed
         oTVI:imageIndex          := IDI_Closed
         oTVI:selectedImageIndex := IDI_Closed
      ELSE
         oTVI:imageIndex          := ;
         oTVI:selectedImageIndex := ;
            IDI_Open
      ENDIF

      oEvent:control:setItemAttributes (oTVI)
   ENDIF
```

The call of a method like :FillTreeView (oTVI) would thus have to take place in the ELSE branch of the above IF query. So that a TVI can be detected as node even if it is unfilled, it thus possesses the “expand bit-map” in form of a plus sign, that should produce a dummy TVI with oTVI:value == NIL and is recognizable as dummy over the value query on NIL. This would have to be deleted then with the “correct” filling of the TVI.

The following method FillTreeViewABC() equips a tree with the letters “A” to “Z” and provides those letters with an empty NIL TreeViewItem, whose search is successful in the author file. There is for example an author, whose surname begins with an “E” receives this TVI item and thus a plus button.

```
Method FillTreeViewABC () Class winSplit
   Local symName As Symbol
   Local oTVI As TreeViewItem
   Local oDbTitel As dbTitel
   Local i As DWord
   Local cChar, cOrder As String

   oTVI := TreeViewItem { ;
      #_Root, ;
      “Autoren & ihre Bücher”, ;
      NIL, ;
      IDI_Root, ;
      IDI_Root }
   _oTrvAutor:addItem (#Root, oTVI)
   FOR i:=1 UPTO 26
      oTVI := TreeViewItem { ;
```

```
        String2Symbol (Chr (64+i)), ;
        Chr (64+i), ;
        NIL, ;
        IDI_Closed, ;
        IDI_Closed }
     oTVI:stateImageIndex := 0
     _oTrvAutor:addItem (#_Root, oTVI)
     IF _oDbAutor:seek (Chr (64+i))
        _oTrvAutor:addItem ( ;
        String2Symbol (Chr (64+i)), ;
           TreeViewItem { ;
        String2Symbol (Chr (i)), ;
        "", NIL })
     ENDIF
  NEXT i
```

As one can see, there is a special meaning to the value of oTVI:value of a TreeViewItem. With a database TreeView control, whose items are filled from a DBF, it would be useful to for example deposit the recno of the record there. The query for NIL would then identify such nodes and the highest root element, that does not possess correspoan equivalent in the DBF.

The above method for reading a node could then look the following way:

```
Method FillTreeView (oTVI_Autor) Class winSplit
  Local cSchl As String
  Local oTVI  As TreeViewItem

  IF IsNumeric (oTVI_Autor:value) ;
     .AND. oTVI_Autor:value > 0
     _oDbAutor:recno := oTVI_Autor:value
     cSchl := _oDbAutor:fieldGet (#SCHL_AUTOR)
     IF _oDbTitel:seek (cSchl)
        // Licht ausschalten
        LockWindowUpdate (oDCtrvTitel: ;
           handle ())
        // Dummy löschen
        oTVI := oDCtrvAutor:getFirstChildItem ( ;
           oTVI_Autor:nameSym)
        IF oTVI != NULL_OBJECT
           oDCtrvAutor:deleteItem (oTVI)
        ENDIF
        // Alle Titel des Autors einlesen
        DO WHILE .NOT. _oDbTitel:eof ;
           .AND. _oDbTitel:fieldGet ( ;
           #SCHL_AUTOR) == cSchl
           oTVI := TreeViewItem { ;
              String2Symbol ("TITEL_" + ;
              NTrim (_oDbTitel:recno)), ;
              Trim (_oDbTitel:fieldGet ( ;
              #TITEL)), ;
              NIL }
           oDCtrvTitel:addItem ( ;
              oTVI_Autor:nameSym, oTVI)
            _oDbTitel:skip (1)
         ENDDO
         // Licht wieder einschalten
         LockWindowUpdate (NULL_PTR)
      ENDIF
   ENDIF
```

An own value class for oTVI:value

Frequently it is not sufficient however to assign a number or a simple data type to this value. Clipper 5-Experts (genuine, no semi-Summer-87-programers...) will think back to the progressive achievements and possibilities of the TBColumn-cargo-slots... As a VO programmer one here uses its own class, whose instances exeptionally may even be of the type "export".

Such a class is not inherited; the export instances are permitted in as much as that the necessity for verification of the code, in order to prevent for example an incorrect overwriting of values necessarily does not exist. Here the class only has the sense to package different values and to organize it through only one single object, that is stored in the oTVI:value.

In the case of the user-right trees shown in the figure the appropriate control in each TVI, the window, on that the red marking framework is drawn (that does not necessarily have to be the owner of the control; for instance with split- or tabcontrols), the current marking status and the boundingbox, thus the square that has to be updated.

```
Class TreeViewItemUserRights
  Export Control      As Control
  Export Marked       As Logic
  Export Window       As Window
  Export BoundingBox As BoundingBox
```

Such a class for the packaging of data does not only correspond a lot to the object-oriented thinking, but is actually also very problem and also "environmentally" compatible: In the daily life we group and organize sections to a whole one; nobody would have the strange idea to move with all things not packed & put into cartons —therefore the transfer of a dozen parameters to a function should also appear quite "unnatural"... Such classes can also be therefore much more complex, as a further example will show:

```
Class TreeItem
  Export NameSym      As Symbol
  Export Caption      As String
  Export Type         As Symbol
  Export ImageIndex  As DWord
  Export Parent       As Symbol
  Export Value        As Usual
  Export ValueType    As String
  Export ReturnBlock As _Codeblock
  Export EditClass    As Symbol
  Export EditWidth    As DWord
  Export EditMode     As DWord
  Export Picture      As String
  Export Disabled     As Logic

  // Für #ListView-Control:
  Export Columns      As Array
  Export ServerName  As Symbol
  Export ValueColumn As Symbol
```

The following class describes the data of a setup object, whose values determine the appearance of a calendar. One can surely estimate quite well the functionality to be expected in some sections of that instance variable. Additionally this class has some protect instance variables that can only be accessed over appropriate methods. That is also very usual with such data packages. Altogether all values are then available through one object that can be easily passed on for example. With further options, that frequently become necessary with extensions, nothing else is to be adapted except for the class CalendarSetup itself.

```
Class CalendarSetup
   Export MarginLeft         As LongInt
   Export MarginTop          As LongInt
   Export MarginRight        As LongInt
   Export MarginBottom       As LongInt

   Export HeightSpace        As LongInt
   Export HeightHeader       As LongInt
   Export HeightDayLine      As LongInt
   Export HeightFooter       As LongInt
   Export HeightDescription As LongInt

   Export WidthWeekLine      As LongInt

   Export AlignFooter        As DWord
   Export AlignDescription  As DWord

   Export BrushBackgroundFocused As Ptr
   Export BrushBackground   As Ptr
   Export BrushFocused       As Ptr
   Export BrushHeader        As Ptr
   Export BrushFooter        As Ptr
   Export BrushDescription  As Ptr

   Export PenBlack           As Ptr
   Export PenGray            As Ptr
   Export PenFocused         As Ptr

   Export FontName           As String
   Export FontDays           As Ptr
   Export FontHeader         As Ptr
   Export FontFooter         As Ptr
   Export FontDescription   As Ptr

   Export ShowHeader         As Logic
   Export ShowFooter         As Logic
   Export ShowDescription   As Logic

   Export MarkWeekend        As Logic
   Export IsHeaderActive     As Logic
   Export IsFooterActive     As Logic
   Export IsDescriptionActive As Logic

   Export FirstDayOfWeek     As DWord

   Protect _oIni             As IniFile
   Protect _aColor           As Array
   Protect _aBrush           As Array
   Protect _aDays            As Array

Access Days
Method Axit ()
Method GetBrush (nItem)
Method GetColor (nItem)
Method Init ()
Method ReadBrushesFromIni ()
Method ReadColorsFromIni ()
Method ReadDaysFromIni ()
```

A test for unique TVI names

The most popular and most persistent error when filling TreeView controls is the inadvertent assignment of not unique names of the TreeViewItem objects. The standard display is usually not even influenced negatively by that; but the expanding and collapsing of the nodes still functions correctly. Only for example while searching in each case the first hit gets selected.

A simple test is the recursive method:ExpandAll(), that expands all nodes (remember the classical retaining problem...). If it does not end, this is an unfailing sign that within a node at least two TreeViewItem lines carry the same oTVI:nameSym name, that is transferred to the init method as the first parameter.

You can find the method as a method of the base class trvStandard in the module "_trvStandard" of the current CAVO2IVO.AEF, that already was attached to the last issue. You can likewise find Methods for the output of all nodes and to expand all nodes there; They are so useful that they fit quite well to the base class. Thus they are available for testing purposes at any time. Similar to the internal VO methods of the class library, their names begin with two underlines.

Editing of a TreeViewItem

Editing a TVI similarly occurs as described above with the ListView control. The event method TreeViewItemEdit (oEvent) of the Window class receives an object of the class TreeViewEditEvent as a parameter that has again the necessary interfaces, that permits the query for the new text. Also here afterwards the value is to be written back into the TreeViewItem.

Own controls in the TreeView

It is more comfortable to produce custom controls. Like that it will be usually more useful, to insert one's own SingleLineEdit in a suitable place to allow the editing of the label with double-clicks instead of using the onboard utility (thus exactly over the text of the Caption of the TreeViewItem). That is actually less difficult, than it may perhaps seem at first sight. Thus a technique is at our disposal, that permits any controls for the modification of TreeViewItems. Finally we cannot only exactly configure the characteristics of the SLE class for the desired purpose, but also insert a combobox for instance.

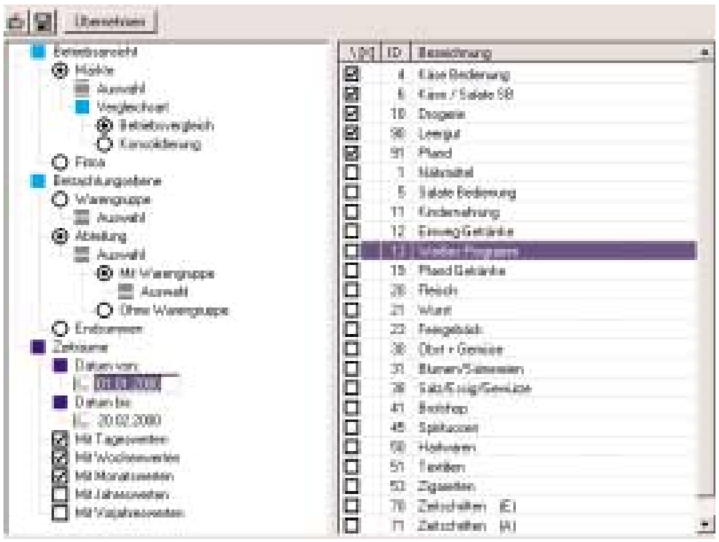


Fig. 8: Edit a TreeViewItems through a dynamic SLE control

However, for your information, the method :GetItemBoundingBox() that returns the object of the class BoundingBox, thus a square type, at the position of a TVI is incorrect. The developer mistook the coordinates of the values for "left" and "bottom" at the selection. Since we introduced our own base class trvStandard anyways (in the CAVO2IVO), that is derived from TreeView (and since VO fortunately is not calledVisual Functions), the modification is not very dramatical.

```
Method GetItemBoundingBox (symItem, lTextOnly) ;
   Class trvStandard
   // Korrigierte Fassung der
   // Original-Methode
   Local hItem      As Ptr
   Local strucRect Is _winRect
   Local oOrigin    As Point
   Local oSize      As Dimension

   Default (@lTextOnly, false)

   hItem := self:__GetHandleFromSymbol (symItem)
   strucRect.left := long (_cast, hItem)
   IF logic (_cast, SendMessage ( ;
      self:Handle(), TVM_GETITEMRECT,  ;
      dword(_cast, lTextOnly), ;
      long(_cast, @strucRect)))
      // BUG 2.5a: left und bottom
      // vertauscht *****************
      oOrigin := Point { strucRect.left, ;
         strucRect.bottom }
      oOrigin := __WCConvertPoint (self, oOrigin)
      oSize   := Dimension{ ;
         strucRect.right - strucRect.left, ;
         strucRect.bottom - strucRect.top}
      Return BoundingBox {oOrigin, oSize}
   ENDIF
   Return BoundingBox {}
```

Thus we can already determine the coordinates over the following line, necessary for a dynamic SingleLineEdit; this is to exactly take the position of the TVI label. However still some instance variables are necessary for the window class. Thus we need _oControl, in that the produced control is stored. Over the flag _lEditControlIsActive it can be queried whether the control is still active. The inquiry on query on _oControl! = NULL_OBJECT is not so reliable as we will later see. The instance variable _oTV contains the TreeViewControl.

```
Class winAuswahl Inherit SplitWindow
   Protect _oFont     As Font
   Protect _oTV       As TreeViewAuswahl

   Protect _oControl As Control

   // Notwendiges Flag, damit auch bei
   // einem Doppelklick auf ein
   // SingleLineEdit das SLE _oControl
   // fokussiert wird. Siehe
   // Dispatcher der Klasse TreeView.
   Protect _lEditControlIsActive As Logic
```

Function GetAveCharWidth()

Additionally we will need a function that determines the average width of a character. After all the width of the desired (to be created dynamically) SLE control is to be indicated both in pixels and in characters.

```
Function GetAveCharWidth ( ;
   oControl As Control) As DWord
   // Ermittelt die durchschnittliche
   // Breite eines Zeichens
   // in einem Control.
   Local tm  Is _winTextMetric
   Local hDC As Ptr

   hDC := GetDC (oControl:handle ())
   GetTextMetrics (hDC, @tm)
   ReleaseDC (oControl:handle (), hDC)

   Return tm.tmAveCharWidth
```

Click into emptiness

With active SLE control one must naturally intercept some possible actions of the user. He could click outside of the SLE, or take away TreeView control over the scrollbars. Up-to-date equipped users could naturally also have the idea to turn the mouse wheel with active SLE. All these actions must store the SLE back into the TreeViewItem (if one would accept the value) and destroy the SLE. This method is to be written for the window (that controls the SLE in form of the instance variables _oControl) and it is very simple:

```
Method DestroyEditControl ()Class winAuswahl
   // Durch Doppeklick etc. erzeugtes
   // SingleLineEdit wieder zerstören.
   IF _oControl != NULL_OBJECT
      _oControl:destroy ()
      _oControl := NULL_OBJECT

      IF .NOT. InCollect ()
         CollectForced ()
      ENDIF
   ENDIF
```

The beginning of the above situations is naturally much more difficult. We need its own TreeView class in each case, so that we can react on the appropriate events. We have to write a new method GetItemBoundingBox() anyways to correct the error of the 2.5-Version. Since doubleclicking on the TVI creates the SLE (by the method still the one that can be shown:CreateEditControl()), it is however not focused. We guarantee the activation in the dispatcher —and ensure for the fact that that does not occur a second time by setting the flag. Here thus the characteristics of the class trvAuswahl:

```
Class trvAuswahl Inherit trvStandard

Method VerticalScroll (oEvent) Class trvAuswahl
   IF self:owner:EditControl != ;
      NULL_OBJECT

      self:owner:SaveEditControl ()
      self:owner:DestroyEditControl ()
   ENDIF

   super:verticalScroll (oEvent)

Method Dispatch (oEvent) ;
   Class trvAuswahl
   // Ansonsten ist das SLE nach einem
   // Doppelklick in den Tree das SLE
   // nicht aktiv zu kriegen; nach dem
   // Doppelklick bleibt der Focus auf
   // dem Tree (was man auch daran
   // erkennen kann, daß die Tree-Zeile
   // blau bleibt).
   // Abfrage auf self:owner:EditControl
   // != NULL_OBJECT und GetFocus () !=
   // self:owner:EditControl:handle ()
   // funktioniert nicht zuverlässig.
   // Besser ein eigenes Flag, daß man
   // eindeutig zurücksetzen kann.
   DO CASE
   CASE self:owner:EditControlIsActive
      self:owner:EditControlIsActive := FALSE
      self:owner:EditControl:setFocus ()
   ENDCASE

   Return super:dispatch (oEvent)
```

Obviously we must be able to access some protect instance variables of the window class winAuswahl from methods of the trvAuswahl class. Since protect variables —nomen est omen; or packaging is everything —do not permit this, we need some interfaces:

```
Access EditControl Class winAuswahl
   // Schnittstelle für den Zugriff auf das
   // Edit-Control auch von trvAuswahl aus.
   Return _oControl

Access EditControlIsActive Class winAuswahl
   // Schnittstelle für den Workaround in
   // trvAuswahl. Siehe Method Dispatch ()
   // Class trvAuswahl
   Return _lEditControlIsActive

Assign EditControlIsActive (lActive) ;
   Class winAuswahl
   Return _lEditControlIsActive := lActive
```

SingleLineEdit class sleTreeView

Naturally also certain events in the SingleLineEdit control itself must be intercepted. So for instance the keys VK_Return, VK_Up, VK_Down and VK_Tab have to close the SLE and write back the value. Pushing the Esc key terminates the input without saving. Thus we also need your own class here:

```
Class sleTreeView Inherit SingleLineEdit

Method Dispatch (oEvent) Class sleTreeView
   DO CASE
   CASE oEvent:message == WM_KeyDown
      // Ausgelöst werden kann das SLE
      // auch über TreeViewKeyDown ()
      // (TreeViewKeyUp () gibt es nicht);
      // daher ist die Taste noch gedrückt,
      // wenn das SLE aufgebaut wird. Damit
      // es nicht gleich
      // wieder geschlossen wird, muß hier
      // WM_KeyDown abgefragt werden.
      DO CASE
      CASE InList (oEvent:wParam, ;
         VK_Return, VK_Up, VK_Down, VK_Tab)
         self:owner:owner:SaveEditControl ()
         self:owner:owner:DestroyEditControl ()
         Return 1

      CASE oEvent:wParam == VK_Escape
         self:owner:owner:DestroyEditControl ()
         Return 1
      ENDCASE
   ENDCASE

   Return super:dispatch (oEvent)
```

Write SLE control back into the TreeViewItem

When storing the SLE control back in the TVI naturally a type adjustment would be possible. In the concrete case the oTVI:value value is an object of the above class TreeItem, that contains for example also an export instance variable:ValueType with the type of the SLE control. In the simplified case naturally the text input in the SLE can be written back over oTVI:textValue := _oControl:textValue into the TreeViewItem. You should however never forget in no case an updating of the TreeView control: _oTV:setItemAttributes (oTVI).

```
Method SaveEditControl (oTVI) CLass winAuswahl
   // Wert des SingleLineEdits in das
   // TreeVieItem übertragen.

   IF IsNil (oTVI)
      oTVI := _oTV:getSelectedItem ()
   ENDIF
   IF oTVI != NULL_OBJECT
      oTVI:textValue := _oControl:textValue
      _oTV:setItemAttributes (oTVI)
   ENDIF
```

Trigger of the SLE production

That dynamic SingleLineEdit control can alternatively be produced with one doubleclick on a TreeViewItem or when selecting a TVI. You will find both (very managable) versions following next:

```
Method TreeViewMouseButtonDoubleClick ( ;
   oEvent) Class winAuswahl
   // Wird aufgerufen bei Doppelklick im
   // TreeView.
   Local oTVI As TreeViewItem

   super:treeViewMouseButtonDoubleClick (oEvent)

   oTVI := oEvent:treeViewItem

   IF oEvent:isLeftButton ;
      .AND. oTVI != NULL_OBJECT
         self:CreateEditControl (oTVI)
   ENDIF

Method TreeViewSelectionChanged (oEvent) ;
   Class winAuswahl
   // Zeile im TreeView-Control wurde
   // geändert. Bei Auswahl einer neuen
   // Zeile im Tree bei geöffnetem
   // Edit-Control ist dieses zu speichern
   // und zu zerstören.
   Local oTVI As TreeViewItem

   super:treeViewSelectionChanged (oEvent)

   oTVI := oEvent:oldTreeViewItem

   IF oTVI != NULL_OBJECT
      IF _oControl != NULL_OBJECT
         self:SaveEditControl (oTVI)
         self:DestroyEditControl ()
      ENDIF
   ENDIF

   oTVI := oEvent:newTreeViewItem

   IF oTVI != NULL_OBJECT
      self:CreateEditControl (oTVI)
   ENDIF
   // geändert. Bei Auswahl einer neuen
   // Zeile im Tree bei geöffnetem
   // Edit-Control ist dieses zu speichern
   // und zu zerstören.
   Local oTVI As TreeViewItem

   super:treeViewSelectionChanged (oEvent)
   oTVI := oEvent:oldTreeViewItem
   IF oTVI != NULL_OBJECT
      IF _oControl != NULL_OBJECT
         self:SaveEditControl (oTVI)
         self:DestroyEditControl ()
      ENDIF
   ENDIF
   oTVI := oEvent:newTreeViewItem
   IF oTVI != NULL_OBJECT
      self:CreateEditControl (oTVI)
   ENDIF
```

Now finally: producing SLE on a TreeView control

Now we groped our way sufficiently careful towards the actual method. Finally all accessory entities are available and we can turn towards the method CreateEditControl(), that creates an SLE at the position of the doubleclicked or selected TreeViewItem. The position and size of the SLE determine the boundingbox of the TVI; the TreeView control is the owner of the control. Besides the (neutral) ID –1 the usual SLE styles are transferred.

```
Method CreateEditControl (oTVI) ;
   Class winAuswahl
   // Erzeugt in _oControl ein Control der
   // Klasse sleTreeView. Das Control muß
   // aber nicht unbedingt ein SLE sein!
   // Denkbar wäre auch eine Combobox
   // oder dergleichen.
   Local oBoundingBox As BoundingBox
   Local nLeft, nBottom, nWidth, ;
         nHeight As DWord
   Local symClass As Symbol

   // Sollte ein Control noch da
   // sein: Sauber zerstören.
   self:DestroyEditControl ()

   // Bug in 2.5a: Korrigierte Methode!
   oBoundingBox := ;
      _oTV:GetItemBoundingBox ( ;
      oTVI:nameSym, TRUE)

   nHeight := 20
   nLeft   := oBoundingBox:origin:x - 2
   nBottom := _oTV:origin:y + ;
              oBoundingBox:origin:y – 2
   nWidth  := oBoundingBox:width

   symClass  := #sleTreeView

   _oControl := CreateInstance ( ;
      symClass, ;
      _oTV, ;
      -1, ;
      Point { nLeft, nBottom }, ;
      Dimension { nWidth, nHeight }, ;
      WS_Child + WS_TabStop + ;
      ES_AutoHScroll)

      _oControl:value := oTVI:textValue
      _oControl:show ()
```

```
      // Workaround-Flag; siehe
      // Dispatch () Klasse trvAuwahl
      _lEditControlIsActive := TRUE
   ENDIF
```

Programming of a Wheel Mouse

Not only in the sourcecode editor of VO the wheel is used now (to try it, if you were just thinking how to do: Ctrl+mousekheel). A wheel mouse normally functions automatically for Windows controls with Scrollbars that allow to leaf through; but for instance when increasing/reducing items, with CustomControls users think of them as useful. Also one should not be frightened of combinations with SHIFT /Ctrl /Alt key.

With the described production of a dynamic SLE control for editing a TreeViewItems a wheel movement should close a possibly active SLE and write its value back accordingly. The above dispatcher of the TreeView class trvAuswahl is to be supplemented by only a few lines of code. Window message WM_MouseWheel is (still?) not defined in VO 2.5. Such values can be determined in addition, marvelously by the VO example “Private Eye” (that is a very comfortable and extremely useful tool for spying Windows applications and windows).

If WM_MouseWheel arrives in the dispatcher of a window or a control over the oEvent:message message, the wheel was operated. If the wheel was turned upward, the HiWord value oEvent:wParam is –120, otherwise 120. The direction of rotation plays no role for the trvAuswahl example, but it can however be queried otherwise.

```
Define WM_MouseWheel := 0x020A

Method Dispatch (oEvent) Class trvAuswahl
   CASE oEvent:message == WM_MouseWheel ;
      .AND. self:owner:EditControl != ;
      NULL_OBJECT
      // HiWord (wParam) == 120 (Drehung
      // nach unten) bzw. -120 (nach oben)
      self:owner:SaveEditControl ()
      self:owner:DestroyEditControl ()
   ENDCASE
   Return super:dispatch (oEvent)
```

Finally still another example, how one could also test combinations with the SHIFT /Ctrl /Alt key and the mouse wheel. I do not like to use latter version, because after releasing the Alt key the menu of a window is focused. The other examples control the leafing possibilities of a calendar. Note that the negative value –120 doesn’t gets along with the comparison of the HiWord (oEvent:wParam) and must therefore be casted to a word.

```
Method HandleMouseWheelMessage (oEvent) ;
   Class ccCalendarStandard
   //         WM_MouseWheel -> Wochenweise scrollen
   // Shift + WM_MouseWheel -> Tageweise scrollen
   //  Strg + WM_MouseWheel -> Monatsweise scrollen
   //   Alt + WM_MouseWheel ->

   DO CASE
   // Alt+Drehung nach oben ->
   CASE HiWord (oEvent:wParam) == 120 ;
      .AND. GetKeyState (VK_Menu) < 0
   // Alt+Drehung nach unten
   CASE HiWord (oEvent:wParam) == ;
```

```
      Word (_Cast, -120) ;
      .AND. GetKeyState (VK_Menu) < 0
   CASE GetKeyState (VK_Menu) < 0
      // do nothing: Alle anderen
      // Alt+Rad abfangen
   // Shift+Drehung nach oben ->
   CASE HiWord (oEvent:wParam) == 120 ;
      .AND. GetKeyState (VK_Shift) < 0
      self:SkipDays (-1)
   // Shift+Drehung nach unten
   CASE HiWord (oEvent:wParam) == ;
      Word (_Cast, -120) ;
      .AND. GetKeyState (VK_Shift) < 0
      self:SkipDays (+1)
   CASE GetKeyState (VK_Shift) < 0
      // do nothing: Alle anderen
      // Shift+Rad abfangen
   // Strg+Drehung nach oben ->
   CASE HiWord (oEvent:wParam) == 120 ;
      .AND. GetKeyState (VK_Control) < 0
      self:SkipMonths (-1)
   // Strg+Drehung nach unten
   CASE HiWord (oEvent:wParam) == ;
      Word (_Cast, -120) ;
      .AND. GetKeyState (VK_Control) < 0
      self:SkipMonths (+1)
   CASE GetKeyState (VK_Control) < 0
      // do nothing: Alle anderen
      // Strg+Rad abfangen
   // Drehung nach oben ->
   CASE HiWord (oEvent:wParam) == 120
      self:SkipDays (-7)
   // Drehung nach unten
   CASE HiWord (oEvent:wParam) == ;
      Word (_Cast, -120)
      self:SkipDays (+7)
   ENDCASE
   Return 1
```

Before I forget...

A few little things between the lines, that I would not like to conceal, should close the article. Small deviations from the otherwise always extremely severe topic timetable are permitted here.

Pallet window title

Who would like to perhaps once use those narrow window title borders, like one can frequently find in pallet windows —the window editor of VO uses them for example —, does not possibly suspect at all that they are naturally available in VO applications. However they can be assigned not in the window editor, but require little manual work, for example in the init method of a window: or naturally directly as method of a window base class.

```
Method SetPaletteStyle () Class dlgWindow
   // Paletten-Caption; schmaler und mit
   // nicht-fettem Systemfont.
   // Transparenter Balken unterhalb der
   // Caption muß neu gezeichnet werden.
   SetWindowLong (self:handle (),  GWL_ExStyle, ;
      _Or (GetWindowLong (self:handle (), ;
      GWL_ExStyle), WS_Ex_PaletteWindow))

   SetWindowPos (self:handle (),  NULL_PTR, 0, 0, ;
      self:size:width, self:size:height, ;
      _Or (SWP_NoMove, SWP_DrawFrame))
```

As one can see, the redrawing of the area underneath the title border is necessary -this area will otherwise be transparent and lets the background gleam through unpleasantly when setting the WS_Ex_PaletteWindow style. The manifold universal API function SetWindowPos() —with it controls can also be shifted or for instance be provided with a new tab order— here provides the necessary "view density".

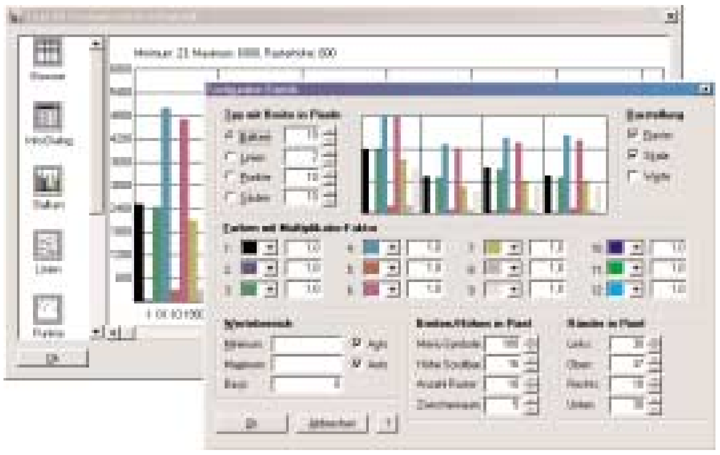


Fig. 9: Dialog with pallet window style

Right-justified SingleLineEdit controls for VO 2.0, too

That in former times needed still another "smart control" class, whose palpation jerks and displays one had to build in laborious small and manual work oneself, is finally generally available since Windows 98/NT ff: SingleLineEdit control with right-justified display. I gladly used a MultiLineEdit control trimmed on single-lineness, that already possessed this style - VO 2.5 users finally find the style "ExAlignment=Right" on the latch "ExStyles" in the window editor that is ideal for numbers in particular.

VO 2.0 users do not have to grief here —a renouncement of transfer to the current version is hereby however by any means not talked about... The new style WS_EX_RIGHT cannot be assigned however dynamically, thus subsequently approximately with oSleControl:setExStyle (WS_EX_RIGHT). The possibility of attaching it to the generated resources entity is possible, but rather frustrating: you may repeat that then with each new generation of the code...

```
   CONTROL   "",  TABKARTEISTAMM_VO_NACHNAME, ;
     "Edit"     ,  ES_AUTOHSCROLL|WS_TABSTOP| ;
     WS_CHILD|WS_BORDER,   64, 20, 134, 12, ;
     WS_EX_CLIENTEDGE|WS_EX_RIGHT
```

In addition, the new style can be built easily into the available CAVOWED.INF of the version 2.0. With the SingleLineEdit properties on the latch "ExStyles", "ExAlignment" adjustment can then be selected between "left" and "right". If you look at the generated window resources with activated "right" switch, then you will discover the new style WS_EX_RIGHT at the end of the SLE control.

Add the following two lines in addition to the sections "[controls]" and "[StdProperties]". In the line "ExStyles", attach the new variable "ExAlignment" and receive a further Combobox with the two values "Left" and "right". Backing up the old CAVOWED.INF does not harm at all... The lines naturally are to be entered without word wrap.

```
[CONTROL]
ExWindowStyle08=ExAlignment,WS_EX_LEFT:
   WS_EX_RIGHT(EXALIGNMENT)

[StdProperties]
EXALIGNMENT=Left,Right
...
ExStyles=(Extended Window Styles)
   Clip Siblings, Right-To-Left Reading,
   No Parent Notify,Accept Files,
   Transparent,Client Edge,Static Edge,
   Modal Frame,ExAlignment
```

Variable resources path

A Screenshot in the last output already revealed it; but, the possibility of specifying the path of resources as for instance a bitmap icon or a cursor over a variable seems to have not yet gotten around very much. The advantage —particularly together with the renouncement of the generated entities —facilitates the exchange with other developers. At all one should keep bitmaps, icons etc. in its own DLL with large applications. In the case of modification of the directory structure then the correction of the resources paths is a thing only one "search and replacing" action.

The variable "%ExecutableDir%" contains indicated directory under file, setup. In the case of a modification however VO is to be left briefly, since it is only covered with the start of the program. How one can see, the relative paths can also be indicated.

```
Resource icoU11 Icon %ExecutableDir%\..\bmp\Krank.ico
```

Outlook and temporary conclusion

No, no fears —in the next issue of the SDT this will naturally continue. Topic will probably then be the most underestimated and the least of all assigned control: or did you already program a CustomControl? You will be surprised, how simple their implementation is: and how beautiful things you can program with it... Briefly said, it is used whenever you like to draw a control completely yourself whenever one of the standard components does not correspond to the desired purpose.

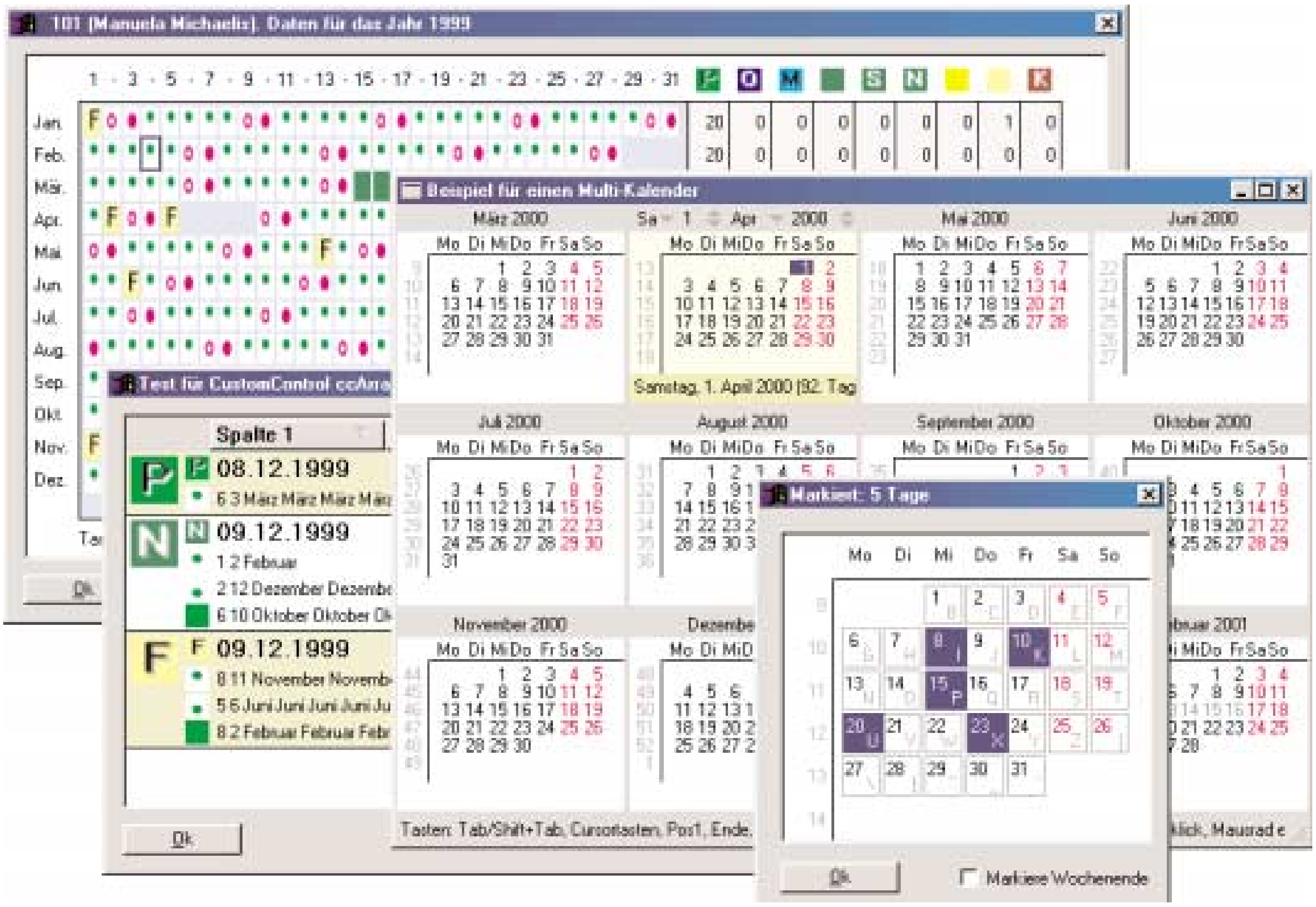
As incentive & luring a few examples of CustomControls: see fig. 10 below...

VO has a completely pleasing example in form of a calendar AxtiveX control. However once more the description of the basic methodology lacks: an inquiry resulted that even expert VO users do not use this useful control, since they do not know, how one starts at all —and don't suspect at how simple it is after all. They will not be able to get along without CustomControls any longer with the release of the next issue of the SDT...

The files on that CD correspond to those from the last issue; the smaller tips can be entered rapidly by manual input or be cut out of the PDF file on the CD. Updating and or other VO examples can be found shortly on my Homepage. Finally: Consider my new EMail address please, my previous CompuServe ID is not valid any more.

Ivo Wessel

EMail: *email@ivo-wessel.de*


101 [Manuela Michaelis]. Daten für das Jahr 1999
Beispiel für einen Multi-Kalender
Test für CustomControl ccAre
Spalte 1
Markiert: 5 Tage
Samstag, 1. April 2000 (92. Tag
Tasten: Tab/Shift+Tab, Cursortasten, Pos1, Ende.
Markiere Wochenende
Ok