Inno Setup Revisited

product review by Craig Murphy

I last wrote about Inno Setup by Jordan Russell (http://www.jrsoftware.org/) in the Sept/Oct 2002 issue of this fine publication. The review itself was called "Creating professional installation using Inno Setup". Now, you may be wondering why, some two years later (almost), I find myself reviewing it again?

Well, I took delivery of a new laptop a couple of months ago – last week I had to make some minor modifications to an application and its associated setup.exe. Naturally, I installed Delphi 6 and a few select components, such that I could re-build all my applications. However, I had forgotten to install Inno Setup, so I could not build the application's setup.exe.

I had been following the Inno Setup versioning, having moved the said application from Inno Setup 2.x through to Inno Setup 3.x. In July 2003, Inno Setup 4.x appeared. Like most of us, I'm sure, I kept using Inno Setup 3.x, because it worked – if it ain't broke, don't fix it. However, March 2004 saw the arrival of a new laptop and with it a new operating system. Call me reckless; I put caution to the wind and installed Inno Setup 4.x.

Apart from making building an application's setup.exe really easy, Inno Setup 4.x has some excellent "new" features, such as the inclusion of Pascal-script (http://www.remobjects.com/?ps). It's this feature that I would like to spend some time looking at, hence Inno Setup Revisited. I'll also be looking at some of the third party tool support for Inno Setup – I won't cover them in too much detail as I'm saving that for a subsequent article/review.

Introduction

There are many excellent commercial installers available, however sometimes their price tag, whilst well worth the investment, can be too much for the lone developer or the small software house. If you cannot afford the commercial offerings, you have three options.

Firstly, you could develop your own – this adds a one-off time/cost to your project and may require project-by-project customisations. You would have to design, develop, test, debug, re-test, etc. Suddenly, what started out as small project, becomes a complete project in its own right. After all, most of us have a hard enough time writing the application that we're trying to distribute via the installer! So, this is probably not the most viable option.

Secondly, you could just bundle all the files that your applications require on to a CD and then prepare a complicated "crib sheet" that contains the instructions required to perform an installation. This is potentially fraught with danger – for example, what if the machine that you are installing your application onto already has a newer version of a particular DLL? Do you overwrite it? Do you ask the user? Do you simply ignore it? These scenarios have to be catered for and catering for them on an application-by-application basis will prove to be a time-consuming process.

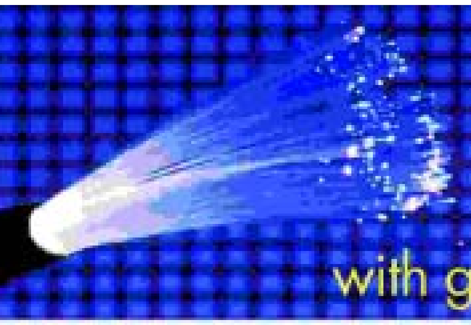
Thirdly, you could opt to use a free/shareware installer. There are many of these available from the various freeware/shareware Internet sites. However, personally speaking, I find that freeware/shareware written using our tools (Delphi, CBuilder, etc.) to be of better quality and usefulness than the freeware/shareware that has been written use other major development environments – they always seem to lack something.

To that end, I use a freeware product called Inno Setup. However, since I have already reviewed Inno Setup (as has fellow DGer, Rob Bracken: The Delphi Magazine, Issue 99, November 2003), I will concentrate on the newer features and will touch on the supporting tools. On a related note, this month's Grey Matter *HardCopy* magazine, Issue 24, June 2004, covers some of the commercial application installers.

Inno Setup

Inno Setup is a freeware product that is free for personal and commercial use – it's even supplied with full source code. Jordan Russell developed the tool itself, with assistance from Martijn Laan. It has been in existence since 1997 and as such enjoys the benefits of being tried and tested.

As an individual I have been using Inno Setup since 1999 and have found it to be particularly fit for purpose: Inno Setup does what it says on the tin. However, prior to 1999, as a company, our tool of choice was InstallShield. So why did I choose to abandon the major player in favour of a freeware equivalent? Well it certainly wasn't because InstallShield lacked the features that my install required, far from it. In November 1999, Developer-friendly web hosting and dedicated servers



with great support

Are you fed up with explaining what you need to your web host? Wish they understood about .NET, ISAPI, CGI and XML? Wish you could get some *real* technical support?

At TDMWeb we know about software development: we also publish *The Delphi Magazine*, and deal with developers all day long!

So talk to us and you're talking with friends. We even have a special Windows Developer package designed just for the development and testing of web applications.

From simple shared hosting to your own dedicated server (complete with our Total Management service if you need it), we can do it all.

- Windows and Linux hosting and dedicated servers.
- ISAPI, ASP, ASP.NET, CGI, PHP, Perl, SSI, MySQL, Access, SQL Server, SOAP, XML, XSLT and more.
- Spam and virus protection.
- Excellent prices, great support.

Full package details at **www.TDMWeb.com**
Call +44 (0)870 740 7610 or Email info@tdmweb.com

TDMWeb

Would you like to have more time for the important things in life?

Would you love to get rid of the BDE?

Do you want to improve your performance and stability?

Are you looking for an easy port to Client/Server?

We have THE solution for you:

NEW: Version 7.0

Advantage Database Server (ADS) – Probably the leanest embedded RDBMS available! Advantage Database Server gives you more time and money for the important things in life...

Advantage Database Server is a true alternative to conventional SQL Client/Server databases – at only a fraction of the cost. ADS already has more than 1.5 million users! See for yourself: www.advantagedatabase.com

ADVANTAGE
DATABASE SERVER

Advantage Database Server offers

- native client kits for Delphi/C++ Builder / Kylix (TDataSet Descendant) including source code
- eliminates need for both BDE & dbExpress
- free Advantage Local Server
- multi platform support (MS Windows, Linux, Novell NetWare)
- easy implementation / integration / installation / administration
- scalability (local, peer-to-peer, client/server, Internet with one set of code)

Extended Systems®

Extended Systems Bristol Ltd
7 - 8 Portland Square
Bristol BS2 8SN
Tel: +44 (0) 117 901 5000
Fax: +44 (0) 117 901 5001
advantage@extendedsystems.co.uk
www.extendedsystems.com

I chose Inno Setup because it allowed me to build a setup.exe that fitted onto a floppy disc – something that I struggled to achieve with the version of InstallShield at the time.

Given the profusion of freeware applications that are available, it's often worth sharing our success stories amongst ourselves – it wouldn't be the first time I've just mentioned something in passing (usually at a DG meeting) only to have another member jump down my throat for not telling him / her sooner! And it works both ways; I've been at meetings where another member casually mentions something that has been niggling me for a while – and they've been sitting on the solution, not realising that it's a useful snippet. *[That's a BIG hint for all readers to send me at least one snippet for the magazine! Ed.]*

Inno Setup Basics

Those of you who read my first review of Inno Setup will know that it uses an .ini file format to represent an installation. This means that out of the box we will need to use a text editor to manage installations. Whilst this may sound like a step backwards from today's fancy GUI installers, the .ini files are so simple and Inno Setup is so well documented, that such files are a delight to work with.

During my first review of Inno Setup I presented a reasonably complete setup.exe or application installer, as they seem to be called today. I demonstrated that Inno Setup has two modes of operation: via a wizard or directly via an ini file. I also explained how Inno Setup could be used to distribute SQL Links: perhaps not much use these days, however back then it was.

Over the course of this review I will be covering a different set of Inno Setup's features. Since I last reviewed Inno Setup I had to achieve three [new] things with my application installer.

Firstly, I needed to deploy one of my existing applications (already packaged up inside a setup.exe) silently. In other words there could be no installation wizards popping up asking the user to click Next to continue, etc. Inno Setup is supplied with a wealth of command-line options, one of which allows silent installation. Here are the four command-line options that I use most frequently:

/SP- Disables the This will install... "Do you wish to continue?" prompt at the beginning of Setup.

/SILENT, **/VERYSILENT** Instructs Setup to be silent or very silent.

/NOCANCEL Prevents the user from cancelling during the installation process, by disabling the Cancel button and ignoring clicks on the close button. Useful along with / SILENT.

Secondly, I had to replace SQL Links with InterBase 6.0.2 (or similar). My first review explained how to deploy the BDE and SQLINT32.DLL via an Inno Setup script. It used something called MiniReg.exe and BdeInst.dll. Apart from the deprecation of SQLLinks, I had other reasons to move the application further towards a stable InterBase footing. I had encountered problems during mass table copy operations using the BDE – I was getting spurious [random] access violations during seemingly normal database operations. The solution was to use IBX instead…and what a performance increase that gave my application!

Thirdly, I had to e-mail my setup.exe to a customer site for them to install on their own. Now, I don't know about your user base, but mine are happy enough being absolute wiz kids with Microsoft Excel (and my application), but ask some of them to rename a file, or drag'n'drop a file using Explorer and they'll stare at you in exactly the same way Manuel in Fawlty Towers did just after he said "¿Que?". Rather than try to explain the concepts of backing files up or renaming the existing files, I decided now is the time to augment the setup.exe such that it renames the existing files before installing the new ones. The application in question is a cost estimating application, therefore the ability to roll-back to the last known to be good version is very useful.

But how did I achieve that third action: renaming files? Well, in my opinion, one of the most significant new features found in Inno Setup is its scripting capability. That's what I plan to focus on for the remainder of this review.

Straight in to Script

Inno Setup 4.x provides us with an event-driven architecture, giving us total control over what our setup script can achieve. Think of any setup.exe that you have looked at recently and you should recall that they are typically wizard-driven with a Next and Back button culminating in a Finish button.

A typical Inno Setup setup wizard has any one of the following pages available: Welcome, License Agreement, Password, Information, User Information, Select Destination Location, Select Components, Select Start Menu Folder, Select Tasks, Ready to Install, Preparing to Install, Installing, Information, Setup Completed.

To help us manage these pages, Inno Setup provides us with access to Next / Back button click events that indicate which page is being displayed. There are also a number of constants associated with the page turning event: wpWelcome, wpLicense, wpPassword, wpInfoBefore, wpUserInfo, wpSelectDir, wpSelectComponents, wpSelectProgramGroup, wpSelectTasks, wpReady, wpPreparing, wpInstalling, wpInfoAfter, wpFinished.

Suppose we wanted to take control of the Next button such that each time it was pressed it instructed the setup wizard to move to the next page. How easy would that be? Gaining this fine-grained control that we might want from a setup script can be achieved by adding a [Code] section to your Inno Setup .iss file.

The [Code] snippet below demonstrates how we can achieve this:

```
[Code]
function NextButtonClick(CurPage: Integer): Boolean;
var sFileName : String;
begin
  case CurPage of
    wpWelcome:
      begin
        sFileName := ExpandConstant('{pf}\EstExpress\')+'ESTEXPRESS.EXE';
        if FileExists(sFileName) then begin
          RenameFile(sFileName, ChangeFileExt(sFileName, '.141'));

          sFileName := ExpandConstant('{pf}\EstExpress\')+'BP_LOOKUP.GDB';

          if FileExists(sFileName) then
            RenameFile(sFileName, ChangeFileExt(sFileName, '.141'));

          MsgBox('Previous version has been renamed!  Please click on OK to continue.',
           mbInformation, MB_OK);
        end;
      end;
  end;

  Result := True;
end;
```

During the compilation of the .iss setup script, Inno Setup verifies that the code inside the [Code] section conforms to the Pascal Script syntax (there's more about Pascal Script later in this review). As you can see through, it's reassuringly similar to another popular programming language!

Hopefully the intent of the code inside NextButtonClick is obvious: when the Next button is clicked on during the Welcome page, the code inside the wpWelcome's begin-end is invoked. What I'm actually doing inside this function isn't rocket science: I'm simply checking for the existence of two files, then renaming them.

What is rather neat is the ExpandConstant function. It takes a string as its parameter, in this case {pf} for Program Files, and expands it to match the local system value for that variable. {pf} on most Windows installations points to C:\Program Files, however it's not something an installation program should assume. Inno Setup recognises this and surfaces a whole host of {} constants that we can use in the setup script.

Indeed if you read through listing 1 you will find reference to {app} – this constant is actually generated as a result of items in the [Setup] section: DefaultDirName. {app}, in listing 1's case, may typically point to C:\Program File\EstExpress.

```
; ESTEXPRESS SETUP.EXE SCRIPT (LITE)

[Setup]
AppName=EstExpress
AppVerName=EstExpress - Estimating System
AppCopyright=Copyright © 1999-2004 Currie & Brown
DefaultDirName={pf}\EstExpress
DefaultGroupName=EstExpress
Uninstallable=1
BackColor=$3B00A8
WizardStyle=modern
InfoBeforeFile=Lite Before.rtf
Compression=zip/9
OutputBaseFilename=setup - lite

[Dirs]
Name: "{app}\XLS"
Name: "{app}\config"
Name: "{app}\images"
```

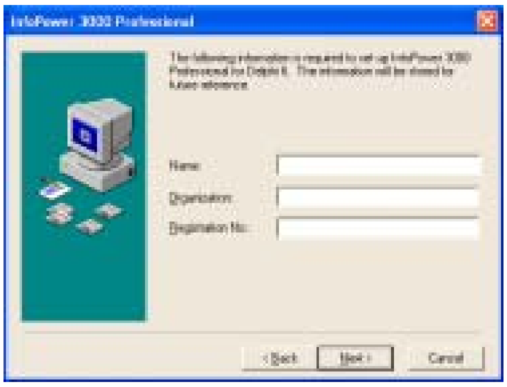
```
[Files]
Source: “EstExpress.EXE”; Flags: ignoreversion; DestDir: “{app}”
Source: “BLANK\BP_LOOKUP.GDB”; DestDir: “{app}”
Source: “BLANK\BP_BLANK.XLT”; DestDir: “{app}\XLS”
Source: “BLANK\REGISTER.XLT”; DestDir: “{app}\XLS”

Source: “images\Scene1.bmp”; DestDir: “{app}\images”
Source: “images\Scene2.bmp”; DestDir: “{app}\images”
Source: “images\Scene3.bmp”; DestDir: “{app}\images”
Source: “images\Scene4.bmp”; DestDir: “{app}\images”
Source: “images\Scene1_256.bmp”; DestDir: “{app}\images”
Source: “images\Scene2_256.bmp”; DestDir: “{app}\images”
Source: “images\Scene3_256.bmp”; DestDir: “{app}\images”
Source: “images\Scene4_256.bmp”; DestDir: “{app}\images”

Source: “config\config.xml”; Flags: ignoreversion; DestDir: “{app}\config”

[UninstallDelete]
Type: filesandordirs; Name: “{app}\XLS”
Type: filesandordirs; Name: “{app}\config”
Type: filesandordirs; Name: “{app}\images”

[Code]
function NextButtonClick(CurPage: Integer): Boolean;
var sFileName : String;
begin
  case CurPage of
    wpWelcome:
      begin
        sFileName := ExpandConstant(‘{pf}\EstExpress\’)+’ESTEXPRESS.EXE’;
        if FileExists(sFileName) then begin
          RenameFile(sFileName, ChangeFileExt(sFileName, ‘.141’));

          sFileName := ExpandConstant(‘{pf}\EstExpress\’)+’BP_LOOKUP.GDB’;

          if FileExists(sFileName) then
            RenameFile(sFileName, ChangeFileExt(sFileName, ‘.141’));

          MsgBox(‘Previous version has been renamed!  Please click on OK to continue.’,
           mbInformation, MB_OK);
        end;
      end;
  end;

  Result := True;
end;
```

Listing 1 – a complete Inno Setup script including a [Code] section

More Control!

Using Inno Setup’s scripting capability we can easily use it to provide us with a means of inserting our own custom pages / screens during the setup wizard’s execution. Those of you familiar with Woll2Woll’s InfoPower or FirstClass component sets will probably recall that their setup.exe records your name, your company name and a serial number.

Figure 1 presents a screenshot of Woll2Woll’s InfoPower setup wizard.



InfoPower 3000 Professional

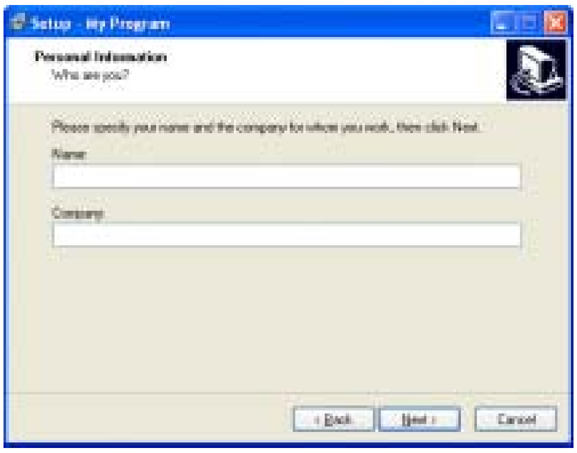
The following information is required to set up InfoPower 3000 Professional for Delphi 6. The information will be stored for future reference.

Name:
Organization:
Registration No:

Figure 1 – Some application installers record personal information

We can replicate that functionality using Inno Setup and copious use of the [Code] section.

What I'm about to go through may appear to be convoluted during the first pass. To make your life a little easier, let's start with what we hope to finish with. Figure 2 presents our desired outcome. Let's go through the steps required to achieve this.


Setup - My Program

Personal Information
Who are you?

Please specify your name and the company for whom you work, then click Next.

Name:

Company:

Figure 2 – Inno Setup let's us record personal information too

Listing 2 presents the entire .iss file that I'll be going through – I've marked up some of the salient bits using a bold font.

```
; — CodeDlg.iss —
;
; This script shows how to insert custom wizard pages into Setup and how to handle
; navigation between those pages if multiple custom pages are inserted after each
; other. Furthermore it shows how to 'communicate' between the [Code] section and
; the regular Inno Setup sections using {code:...} constants and finally it shows
; how to customize the settings text on the 'Ready To Install' page.

[Setup]
AppName=My Program
AppVerName=My Program version 1.5
DefaultDirName={pf}\My Program
DisableProgramGroupPage=yes
UninstallDisplayIcon={app}\MyProg.exe

[Files]
Source: "MyProg.exe"; DestDir: "{app}"
Source: "MyProg.hlp"; DestDir: "{app}"
Source: "Readme.txt"; DestDir: "{app}"; Flags: isreadme

[Registry]
Root: HKCU; Subkey: "Software\My Company"; Flags: uninsdeletekeyifempty
Root: HKCU; Subkey: "Software\My Company\My Program"; Flags: uninsdeletekey
Root: HKCU; Subkey: "Software\My Company\My Program\Settings"; ValueType: string; ValueName:
            "Name"; ValueData: "{code:GetUser|Name}"
Root: HKCU; Subkey: "Software\My Company\My Program\Settings"; ValueType: string; ValueName:
            "Company"; ValueData: "{code:GetUser|Company}"
Root: HKCU; Subkey: "Software\My Company\My Program\Settings"; ValueType: string; ValueName:
            "DataDir"; ValueData: "{code:GetDataDir}"
; etc.

[Dirs]
Name: {code:GetDataDir}; Flags: uninsneveruninstall

[Code]
var
  UserPrompts, UserValues: TArrayOfString;
  UsagePrompts, UsageValues: TArrayOfString;
  Key: String;
  DataDir: String;

function InitializeSetup(): Boolean;
begin
  { Set prompts used on custom wizard pages }
  SetArrayLength(UserPrompts, 2);
  UserPrompts[0] := 'Name:';
  UserPrompts[1] := 'Company:';
  SetArrayLength(UsagePrompts, 3)
  UsagePrompts[0] := ‘Light mode (no ads, limited functionality)’;
  UsagePrompts[1] := ‘Sponsored mode (with ads, full functionality)’;
  UsagePrompts[2] := ‘Paid mode (no ads, full functionality)’;

  { Set default values }
  SetArrayLength(UserValues, 2);
  RegQueryStringValue(HKLM, ‘Software\Microsoft\Windows\CurrentVersion’, ‘RegisteredOwner’,
            UserValues[0]);
  RegQueryStringValue(HKLM, ‘Software\Microsoft\Windows\CurrentVersion’,
            ‘RegisteredOrganization’, UserValues[1]);
  if (UserValues[0] = ‘’) and (UserValues[1] = ‘’) then begin
    RegQueryStringValue(HKCU, ‘Software\Microsoft\MS Setup (ACME)\User Info’, ‘DefName’,
            UserValues[0]);
    RegQueryStringValue(HKCU, ‘Software\Microsoft\MS Setup (ACME)\User Info’, ‘DefCompany’,
            UserValues[1]);
  end;

  SetArrayLength(UsageValues, 3)
  UsageValues[1] := ‘1’;

  { Try to find the settings that were stored last time (also see below). }
  UserValues[0] := GetPreviousData(‘Name’, UserValues[0]);
  UserValues[1] := GetPreviousData(‘Company’, UserValues[1]);
  UsageValues[0] := GetPreviousData(‘Light mode’, UsageValues[0]);
  UsageValues[1] := GetPreviousData(‘Sponsored mode’, UsageValues[1]);
  UsageValues[2] := GetPreviousData(‘Paid mode’, UsageValues[2]);
  DataDir := GetPreviousData(‘DataDir’, ‘’);

  { Let Setup run }
  Result := True;
end;

procedure RegisterPreviousData(PreviousDataKey: Integer);
begin
  { Store the settings so we can restore them next time }
  SetPreviousData(PreviousDataKey, ‘Name’, UserValues[0]);
  SetPreviousData(PreviousDataKey, ‘Company’, UserValues[1]);
  SetPreviousData(PreviousDataKey, ‘Light mode’, UsageValues[0]);
  SetPreviousData(PreviousDataKey, ‘Sponsored mode’, UsageValues[1]);
  SetPreviousData(PreviousDataKey, ‘Paid mode’, UsageValues[2]);
  SetPreviousData(PreviousDataKey, ‘DataDir’, DataDir);
end;

function ScriptDlgPages(CurPage: Integer; BackClicked: Boolean): Boolean;
var
  I, CurSubPage: Integer;
  Next, NextOk: Boolean;
begin
  if (not BackClicked and (CurPage = wpSelectDir)) or (BackClicked and (CurPage = wpReady))
            then begin
    { Insert a custom wizard page between two non custom pages }
    { First open the custom wizard page }
    ScriptDlgPageOpen();
    { Set some captions }
    ScriptDlgPageSetCaption(‘Select Personal Data Directory’);
    ScriptDlgPageSetSubCaption1(‘Where should personal data files be installed?’);
    ScriptDlgPageSetSubCaption2(‘Select the folder you would like Setup to install personal
            data files to, then click Next.’);
    { Initialize the DataDir if necessary }
    if DataDir = ‘’ then
      DataDir := ‘C:\’+UserValues[0];
    { Ask for a dir until the user has entered one or click Back or Cancel }
    Next := InputDir(False, UserValues[0], DataDir);
    while Next and (DataDir = ‘’) do begin
      MsgBox(SetupMessage(msgInvalidPath), mbError, MB_OK);
      Next := InputDir(False, UserValues[0], DataDir);
    end;
    { See NextButtonClick and BackButtonClick: return True if the click should be allowed }
    if not BackClicked then
      Result := Next
    else
      Result := not Next;
    { Close the wizard page. Do a FullRestore only if the click (see above) is not allowed }
    ScriptDlgPageClose(not Result);
  end else if (not BackClicked and (CurPage = wpWelcome)) or (BackClicked and (CurPage =
           wpSelectDir)) then begin
    { Insert multiple custom wizard page between two non custom pages }
    { Now we must handle navigation between the custom pages ourselves }
    { First find out on which page we should start }
    if not BackClicked then
      CurSubPage := 0
    else
      CurSubPage := 2;
    { Then open the custom wizard page }
    ScriptDlgPageOpen();
    { Set the main caption }
    ScriptDlgPageSetCaption('Personal Information');
    { Loop while we are still on a custom page and Setup has not been terminated }
    while (CurSubPage >= 0) and (CurSubPage <= 2) and not Terminated do begin
      case CurSubPage of
        0:
          begin
            { First ask for some user info }
            ScriptDlgPageSetSubCaption1('Who are you?');
            ScriptDlgPageSetSubCaption2('Please specify your name and the company for whom
           you work, then click Next.');
            Next := InputQueryArray(UserPrompts, UserValues);
            if Next then begin
              NextOk := UserValues[0] <> '';
              if not NextOk then
                MsgBox('You must enter your name.', mbError, MB_OK);
            end;
          end;
        1:
          begin
            { Then ask for the usage mode }
            ScriptDlgPageSetSubCaption1('How will you use My Progam?');
            ScriptDlgPageSetSubCaption2('Please specify how you would like to use My Program,
           then click Next.');
            Next := InputOptionArray(UsagePrompts, UsageValues, True, False);
            NextOk := True;
          end;
        2:
          begin
            if UsageValues[0] = '1' then begin
              { Show a message if 'light mode' was chosen above }
              { Skip this page when the user just clicked the Back button }
              ScriptDlgPageSetSubCaption1('How will you use My Progam?');
              Next := OutputMsg('Note: to enjoy all features My Program can offer and to
           support its development, you can switch to sponsored or paid mode at any time by
           selecting ''Usage Mode'' in the ''Help'' menu of My Program after the
           installation has completed.'#13#13'Click Back if you want to change your usage
           mode setting now, or click Next to continue with the installation.', True);
              NextOk := True;
            end else if UsageValues[2] = '1' then begin
              { Ask for a registration key if 'paid mode' was chosen above }
              { This demo accepts only one key: 'isx' (without quotes) }
              ScriptDlgPageSetSubCaption1('What''s your registration key?');
              ScriptDlgPageSetSubCaption2('Please specify your registration key, ' +
           UserValues[0] + '. Click Next to continue. If you don''t have a valid
           registration key, click Back to choose a different usage mode.');
              Next := InputQuery('Registration key:', Key);
              if Next then begin
                { Just to show how OutputProgress works }
                for I := 0 to 10 do begin
                  OutputProgress('Authorizing registration key...', '', I, 10);
                  Sleep(100);
                  if Terminated then
                    Break;
                end;
                NextOk := Key = 'isx';
                if not NextOk and not Terminated then
                  MsgBox('You must enter a valid registration key.', mbError, MB_OK);
              end;
            end;
          end;
      end;
      if Next then begin
        { Go to the next page, but only if the user entered correct information }
        if NextOk then
```

```
          CurSubPage := CurSubPage + 1;
      end else
        CurSubPage := CurSubPage - 1;
    end;
    { See NextButtonClick and BackButtonClick: return True if the click should be allowed }
    if not BackClicked then
      Result := Next
    else
      Result := not Next;
    { Close the wizard page. Do a FullRestore only if the click (see above) is not allowed }
    ScriptDlgPageClose(not Result);
  end else begin
    Result := True;
  end;
end;

function NextButtonClick(CurPage: Integer): Boolean;
begin
  Result := ScriptDlgPages(CurPage, False);
end;

function BackButtonClick(CurPage: Integer): Boolean;
begin
  Result := ScriptDlgPages(CurPage, True);
end;

function UpdateReadyMemo(Space, NewLine, MemoUserInfoInfo, MemoDirInfo, MemoTypeInfo,
           MemoComponentsInfo, MemoGroupInfo, MemoTasksInfo: String): String;
var
  S: String;
begin
  { Fill the ‘Ready Memo’ with the normal settings and the custom settings }
  S := ‘’;
  S := S + ‘Personal Information:’ + NewLine;
  S := S + Space + UserValues[0] + NewLine;
  if UserValues[1] <> ‘’ then
    S := S + Space + UserValues[1] + NewLine;
  S := S + NewLine;

  S := S + ‘Usage Mode:’ + NewLine;
  if UsageValues[0] = ‘1’ then
    S := S + Space + UsagePrompts[0] + NewLine
  else if UsageValues[1] = ‘1’ then
    S := S + Space + UsagePrompts[1] + NewLine
  else
    S := S + Space + UsagePrompts[2] + NewLine;
  S := S + NewLine;

  S := S + MemoDirInfo + NewLine;
  S := S + Space + DataDir + ‘ (personal data files)’ + NewLine;

  Result := S;
end;

function GetUser(S: String): String;
begin
  { Return a user value }
  { Could also be splitted into separate GetUserName and GetUserCompany functions }
  if S = ‘Name’ then
    Result := UserValues[0]
  else if S = ‘Company’ then
    Result := UserValues[1];
end;

function GetDataDir(S: String): String;
begin
  { Return the selected DataDir }
  Result := DataDir;
end;
```

Listing 2 – a complete Inno Setup script including custom wizard pages

I mentioned earlier that Inno Setup provides us with an event-driven architecture and that we received events via such functions as NextButtonClick. Other events fire during the execution of the setup.exe too. For example, there is an initialisation phase that a setup.exe goes through. We can hook into that event by providing an InitializeSetup() function:

```
function InitializeSetup(): Boolean;
begin
  { Set prompts used on custom wizard pages }
  SetArrayLength(UserPrompts, 2);
  UserPrompts[0] := 'Name:';
  UserPrompts[1] := 'Company:';
```

Referring to the code snippet above, we can see that there are two prompts stored in the UserPrompts array. The actual data behind these prompts is going to be stored in the registry using the keys specified in the [Registry] section:

```
Root: HKCU; Subkey: "Software\My Company\My Program\Settings"; ValueType: string; ValueName:
           "Name"; ValueData: "{code:GetUser|Name}"
Root: HKCU; Subkey: "Software\My Company\My Program\Settings"; ValueType: string; ValueName:
           "Company"; ValueData: "{code:GetUser|Company}"
```

What's interesting about the code snippet above the is {code:GetUser|Name} – it ties the registry key entry to the function GetUser:

```
function GetUser(S: String): String;
begin
  { Return a user value }
  { Could also be splitted into separate GetUserName and GetUserCompany functions }
  if S = 'Name' then
    Result := UserValues[0]
  else if S = 'Company' then
    Result := UserValues[1];
end;
```

From the [Registry] section, Inno Setup will essentially call in to the [Code] section to execute the GetUser function, which then sets the value in the registry key. Simple and neat.

We then try to locate previous values for these prompts by looking in the registry:

```
  RegQueryStringValue(HKLM, 'Software\Microsoft\Windows\CurrentVersion', 'RegisteredOwner',
           UserValues[0]);
  RegQueryStringValue(HKLM, 'Software\Microsoft\Windows\CurrentVersion',
           'RegisteredOrganization', UserValues[1]);
  if (UserValues[0] = '') and (UserValues[1] = '') then begin
    RegQueryStringValue(HKCU, 'Software\Microsoft\MS Setup (ACME)\User Info', 'DefName',
           UserValues[0]);
    RegQueryStringValue(HKCU, 'Software\Microsoft\MS Setup (ACME)\User Info', 'DefCompany',
           UserValues[1]);

  { Try to find the settings that were stored last time (also see below). }
  UserValues[0] := GetPreviousData('Name', UserValues[0]);
  UserValues[1] := GetPreviousData('Company', UserValues[1]);

  { Store the settings so we can restore them next time }
  SetPreviousData(PreviousDataKey, 'Name', UserValues[0]);
  SetPreviousData(PreviousDataKey, 'Company', UserValues[1]);
```

After all that set up, now we can see how to create the additional page and give it a caption:

```
    ScriptDlgPageOpen();
    { Set the main caption }
    ScriptDlgPageSetCaption('Personal Information');
```

For each and every page that is displayed the function ScriptDlgPages is called. Its syntax looks like this: function ScriptDlgPages(CurPage: Integer; BackClicked: Boolean): Boolean;

Assuming that we would like this page to be the first page, the following code adds two captions to the page:

```
        0:
          begin
            { First ask for some user info }
            ScriptDlgPageSetSubCaption1('Who are you?');
            ScriptDlgPageSetSubCaption2('Please specify your name and the company for whom
           you work, then click Next.');
            Next := InputQueryArray(UserPrompts, UserValues);
            if Next then begin
              NextOk := UserValues[0] <> '';
              if not NextOk then
                MsgBox('You must enter your name.', mbError, MB_OK);
            end;
          end;
```

The code snippet above also manages the validation of the two items of information: in this case we only need the name field to be filled in correctly.

The remainder of the code presented in listing 2 is, in my opinion, in need of some refactoring – the logic is relatively easy to follow, however it's not the most elegant code…although, if you look at some of my code, the words "pot", "kettle" and "black" spring to mind.

Pascal Script

Inno Setup 4.x saw the inclusion of installer scripting using the Pascal Script scripting language. Pascal Script is part of RemObjects Software product base. It is worth stressing that it is a **free** product.

To quote the RemObjects Software web site:

"Pascal Script is a free scripting engine that allows you to use most of the Object Pascal language within your Delphi projects at runtime. Written completely in Delphi, it's composed of a set of units that can be compiled into your executable, eliminating the need to distribute any external files."

Pascal Script includes the following features: Variables, Constants

Standard language constructs: Begin / End, If / Then / Else, For / To / Downto / Do, Case x Of, Repeat / Until, While, uses, Exit, Continue, Break, Functions (Declared inside or outside of the script)

Standard types: Byte, Shortint, Char, Word, SmallInt, Cardinal, Longint, Integer, String, Real, Double, Single, Extended, Boolean, Array, Record, Enumerations, Variant, IUnknown, IDispatch (dynamic invoke)

Other notable features:

- Ability to import of Delphi functions and classes.
- Assignment of script functions to Delphi events.
- Compilation to a file for later use.
- Easy to use component version.
- Include Files.
- Defines.

Inno Setup has Pascal Script embedded within it. It's remarkably powerful, offering fine-grained control over all aspects of an installation. For example, listing 3 presents the Inno Setup [code] section that allows an Inno Setup setup.exe access to SQL Server via automation.

```
[Code]

{– SQLDMO –}

const
  SQLServerName = 'localhost';
  SQLDMOGrowth_MB = 0;

procedure SQLDMOButtonOnClick(Sender: TObject);
var
  SQLServer, Database, DBFile, LogFile: Variant;
  IDColumn, NameColumn, Table: Variant;
begin
  if MsgBox('Setup will now connect to Microsoft SQL Server ''' + SQLServerName + ''' via a
            trusted connection and create a database. Do you want to continue?',
            mbInformation, mb_YesNo) = idNo then
    Exit;

  { Create the main SQLDMO COM Automation object }

  try
    SQLServer := CreateOleObject('SQLDMO.SQLServer');
  except
    RaiseException(ExceptionType, 'Please install Microsoft SQL server connectivity tools
            first.'#13#13'(Error '''+ExceptionParam+''' occured)');
  end;

  { Connect to the Microsoft SQL Server }

  SQLServer.LoginSecure := True;
  SQLServer.Connect(SQLServerName);
```

```
    MsgBox(‘Connected to Microsoft SQL Server ‘’’ + SQLServerName + ‘’’.’, mbInformation,
              mb_Ok);

    { Setup a database }

    Database := CreateOleObject(‘SQLDMO.Database’);
    Database.Name := ‘Inno Setup’;

    DBFile := CreateOleObject(‘SQLDMO.DBFile’);
    DBFile.Name := ‘ISData1’;
    DBFile.PhysicalName := ‘c:\program files\microsoft sql server\mssql\data\IS.mdf’;
    DBFile.PrimaryFile := True;
    DBFile.FileGrowthType := SQLDMOGrowth_MB;
    DBFile.FileGrowth := 1;

    Database.FileGroups.Item(‘PRIMARY’).DBFiles.Add(DBFile);

    LogFile := CreateOleObject(‘SQLDMO.LogFile’);
    LogFile.Name := ‘ISLog1’;
    LogFile.PhysicalName := ‘c:\program files\microsoft sql server\mssql\data\IS.ldf’;

    Database.TransactionLog.LogFiles.Add(LogFile);

    { Add the database }

    SQLServer.Databases.Add(Database);

    MsgBox(‘Added database ‘’’ + Database.Name + ‘’’.’, mbInformation, mb_Ok);

    { Setup some columns }

    IDColumn := CreateOleObject(‘SQLDMO.Column’);
    IDColumn.Name := ‘id’;
    IDColumn.Datatype := ‘int’;
    IDColumn.Identity := True;
    IDColumn.IdentityIncrement := 1;
    IDColumn.IdentitySeed := 1;
    IDColumn.AllowNulls := False;

    NameColumn := CreateOleObject(‘SQLDMO.Column’);
    NameColumn.Name := ‘name’;
    NameColumn.Datatype := ‘varchar’;
    NameColumn.Length := ‘64’;
    NameColumn.AllowNulls := False;

    { Setup a table }

    Table := CreateOleObject(‘SQLDMO.Table’);
    Table.Name := ‘authors’;
    Table.FileGroup := ‘PRIMARY’;

    { Add the columns and the table }

    Table.Columns.Add(IDColumn);
    Table.Columns.Add(NameColumn);

    Database.Tables.Add(Table);

    MsgBox(‘Added table ‘’’ + Table.Name + ‘’’.’, mbInformation, mb_Ok);
  end;
```

Listing 3 – Automation provides Inno Setup with considerable power

Similarly, listing 4 demonstrates how Inno Setup can talk to Internet Information Server (IIS).

```
  [Code]

  const
    IISServerName = ‘localhost’;
    IISServerNumber = ‘1’;
    IISURL = ‘http://127.0.0.1’;

  procedure IISButtonOnClick(Sender: TObject);
  var
    IIS, WebSite, WebServer, WebRoot, VDir: Variant;
    ErrorCode: Integer;
  begin
```

```
    if MsgBox('Setup will now connect to Microsoft IIS Server ''' + IISServerName + ''' and
              create a virtual directory. Do you want to continue?', mbInformation, mb_YesNo) =
              idNo then
      Exit;

    { Create the main IIS COM Automation object }

    try
      IIS := CreateOleObject('IISNamespace');
    except
      RaiseException(ExceptionType, 'Please install Microsoft IIS first.'#13#13'(Error
              '''+ExceptionParam+''' occured)');
    end;

    { Connect to the IIS server }

    WebSite := IIS.GetObject('IIsWebService', IISServerName + '/w3svc');
    WebServer := WebSite.GetObject('IIsWebServer', IISServerNumber);
    WebRoot := WebServer.GetObject('IIsWebVirtualDir', 'Root');

    { (Re)create a virtual dir }

    try
      WebRoot.Delete('IIsWebVirtualDir', 'innosetup');
      WebRoot.SetInfo();
    except
    end;

    VDir := WebRoot.Create('IIsWebVirtualDir', 'innosetup');
    VDir.AccessRead := True;
    VDir.AppFriendlyName := 'Inno Setup';
    VDir.Path := 'C:\inetpub\innosetup';
    VDir.AppCreate(True);
    VDir.SetInfo();

    MsgBox('Created virtual directory ''' + VDir.Path + '''.', mbInformation, mb_Ok);

    { Write some html and display it }

    if MsgBox('Setup will now write some HTML and display the virtual directory. Do you want to
              continue?', mbInformation, mb_YesNo) = idNo then
      Exit;

    ForceDirectories(VDir.Path);
    SaveStringToFile(VDir.Path + '/index.htm', '<html><body>Inno Setup rocks!</body></html>',
              False);
    if not InstShellExec(IISURL + '/innosetup/index.htm', '', '', SW_SHOWNORMAL, ErrorCode)
              then
      MsgBox('Can''t display the created virtual directory: ''' + SysErrorMessage(ErrorCode) +
              '''.', mbError, mb_Ok);
  end;
```

Listing 4 – Creating virtual directories using IIS

And it wouldn't be me not to mention XML somewhere in an article/review! Listing 5 presents an example of Inno Setup working directly with an XML document using Microsoft's Core XML Services (MSXML).

```
  [Code]

  const
    XMLURL = 'http://cvs.jrsoftware.org/view/*checkout*/ishelp/isxfunc.xml';
    XMLFileName = 'isxfunc.xml';
    XMLFileName2 = 'isxfuncmodified.xml';

  procedure MSXMLButtonOnClick(Sender: TObject);
  var
    XMLHTTP, XMLDoc, NewNode, RootNode: Variant;
    Path: String;
  begin
    if MsgBox('Setup will now use MSXML to download XML file ''' + XMLURL + ''' and save it to
              disk.'#13#13'Setup will then load, modify and save this XML file. Do you want to
              continue?', mbInformation, mb_YesNo) = idNo then
      Exit;

    { Create the main MSXML COM Automation object }
    try
      XMLHTTP := CreateOleObject('MSXML2.ServerXMLHTTP');
    except
      RaiseException(ExceptionType, 'Please install MSXML first.'#13#13'(Error
              '''+ExceptionParam+''' occured)');
    end;

    { Download the XML file }

    XMLHTTP.Open('GET', XMLURL, False);
    XMLHTTP.Send();

    Path := ExpandConstant('{src}\');
    XMLHTTP.responseXML.Save(Path + XMLFileName);

    MsgBox('Downloaded the XML file and saved it as ''' + XMLFileName + '''.', mbInformation,
              mb_Ok);

    { Load the XML File }

    XMLDoc := CreateOleObject('MSXML2.DOMDocument');
    XMLDoc.async := False;
    XMLDoc.resolveExternals := False;
    XMLDoc.load(Path + XMLFileName);
    if XMLDoc.parseError.errorCode <> 0 then
      RaiseException(erCustomError, 'Error on line ' + IntToStr(XMLDoc.parseError.line) + ',
              position ' + IntToStr(XMLDoc.parseError.linepos) + ': ' +
              XMLDoc.parseError.reason);

    MsgBox('Loaded the XML file.', mbInformation, mb_Ok);

    { Modify the XML document }

    NewNode := XMLDoc.createElement('isxdemo');
    RootNode := XMLDoc.documentElement;
    RootNode.appendChild(NewNode);
    RootNode.lastChild.text := 'Hello, World';

    { Save the XML document }

    XMLDoc.Save(Path + XMLFileName2);

    MsgBox('Saved the modified XML as ''' + XMLFileName2 + '''.', mbInformation, mb_Ok);
  end;
```

Listing 5 – Inno Setup and XML

Inno Setup Tools

I am rather content working with Inno Setup's .ini text file format. However, if you are not so happy working at this level there are a couple of supporting tools that might be of interest to you.

Firstly, there is the ISTool (available here: http://www.istool.org/). ISTool wraps up the .ini in an easy to GUI. Secondly, the Inno Setup Form Designer let you design custom forms (wizard pages) using an environment very similar to Delphi's form designer. It's available here: http://isfd.kaju74.de/index.php?id=0,0,0,1,0,0

Conclusions

Inno Setup has been a trusted friend for many years now. Each time I turn to it to help me solve an installer-type problem, it always comes up trumps; it has never let me down. Its feature set goes from strength to strength. Whilst I don't profess to use all the features, I do try to find the time to make myself aware of them – you never know when you might need an element of uniqueness in your setup.exe! Apart from being a fully-featured free tool, Inno Setup is also supplied with source. Delphi source at that. What more could you ask for? Free and with source…amazing.

Next issue, I'll be taking a more in-depth look at another complimentary product (and I mean complimentary in both senses of the word!), Inno Setup Form Designer.

Lastly, I used Microsoft's Virtual PC 2004 (VPC) with Windows XP to run Inno Setup in a clean OS build. I enjoyed considerable success with VMWare, however a recent MSDN purchase provided me with VPC 2004. What a joy VPC is to work with – the clipboard between the virtual WinXp and the real/host WinXP are totally transparent. I can cut'n'paste as if both installations were one and the same. Well, I thought that was pretty cool!

Craig is an author, developer, speaker, project manager and is a Certified ScrumMaster. He specialises in all things XML, particularly SOAP and XSLT. Craig is evangelical about .NET, C#, Test-Driven Development, Extreme Programming, agile methods and Scrum. He can be reached via e-mail at: bug@craigmurphy.com, or via his web site: http://www.craigmurphy.com (where you can also find the source code and PowerPoint files for all of Craig's articles, reviews and presentations).