HOW TO DEVELOP WINDOWS SERVICES

Editor:
Tomáš Mandys, tomas.mandys@2p.cz (2p plus)

Home site:
http://www.2p.cz

Document status:
Version 1.0 First release

Table of Contents

1. Introduction 3
   1.1. NT service 3
   1.2. Console application 3
   1.3. Windows 9x/ME 3
   1.4. Windows NT/2000/XP 3
   1.5. Delphi 4
2. Dual service 4
   2.1. Installation 4
   2.2. Execution 5
   2.3. Status 5
   2.4. SubType 6
3. Source code 6
   3.1. MyDualService.dpr 6
   3.2. dm_Service.pas 19

Disclaimer

1. Introduction

There are many application that running in background and perform some actions, e.g. dial-up host, SMS server, data replicator, web server, etc. Such application does not need any user control, it need just execute. A console application is the most suitable in old *Windows 95/98/ME* system. Good example of such application is port of *Apache* web server for Windows. The Apache is common exe file running in console window. It doesn't need any user input (except `Ctrl+C` to kill) and all the configuration is controlled via command line or configuration file.

The console application is supported also in *Windows NT*, of course. But what is the main difference – multi-user environment. If a machine is dedicated as server where is permanently logged a user, console application is acceptable. If is required better security and transparent behavior, *NT service* is the best solution. Why not, but it requires two different application and two development processes, double debugging, double debugging? No, let's demonstrate how to group console application and *NT service* in one executable.

1.1. NT service

- cannot write anything to standard output (*Write* function), otherwise raises exception
- service is installed into *Service Manager*
- it's impossible obtain application status (except running, paused, stopped) via Service Manager
- Service manager can start, stop, pause and continue service manually or automatically
- service is running under *"LocalSystem"* account unless is explicitly specified other account
- service is running even no user is logged to desktop
- debugging in IDE is difficult

1.2. Console application

- all user output is targeted to console (*OEM* code page, not GUI *ANSI* one)
- only one "resident" instance of application is acceptable
- user can control service via second non-resident instance
- console output is possible
- application appears at taskbar
- help is welcome

1.3. Windows 9x/ME

- *NT service* is not supported

1.4. Windows NT/2000/XP

- there is strong security control

Note that *"service"* is called application supporting dual interface and *"NT service"* is native *Windows NT/2000/XP* service.

1.5. Delphi

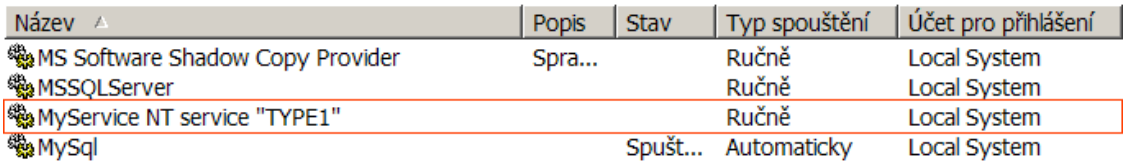
Delphi compiler support developing the *NT services* using *TServiceApplication* and *TService* classes. But *Delphi* approach does not support dual interface and brings very much of overhead. I show how to write lightweight dual interface service application using *Windows API* function. Even example application is written in *Delphi* it is very easy to port to another compiler since only native *API* functions are used.

2. Dual service

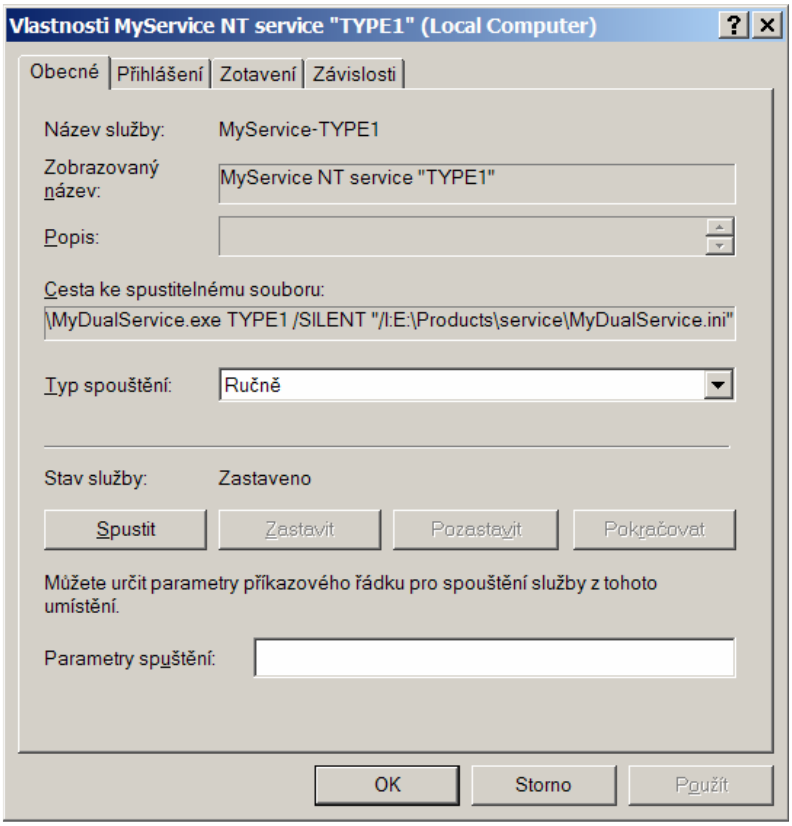
2.1. Installation

The *NT service* must be installed to NT service list managed by the *Service Manager*. Application supports self installation using `/INSTALL` and uninstallation using `/UNINSTALL` switch. If the *NT service* should be executed automatically it necessary change kind of execution in service properties dialog from *Manually* to *Automatically*.

```
mydualservice TYPE1 /INSTALL
```



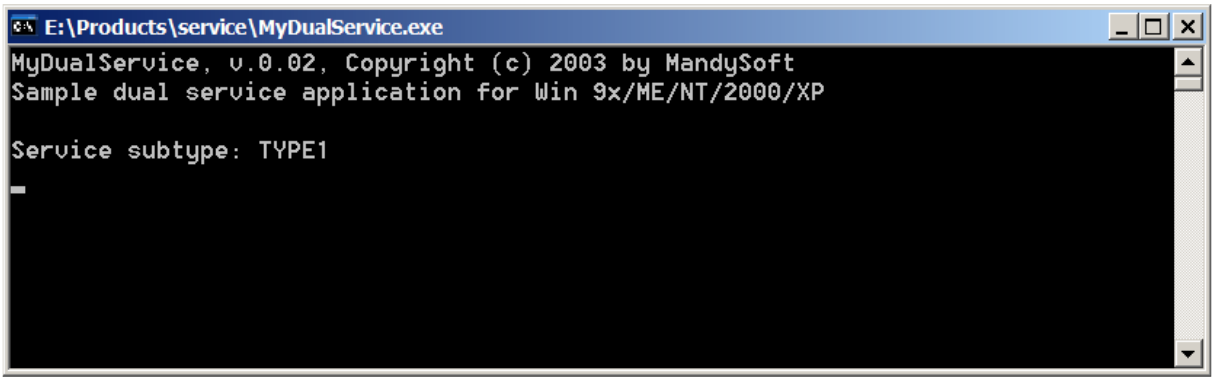
Properties dialog of *MyService NTservice "TYPE1"*. Note that `/SILENT` and `/I:config` switches were automatically added to path.



2.2. Execution

The service application should be running as invisibly as possible without user interaction. It does not support any direct user input and output is redirected to log file (or to console). User can control some behavior using either using the Service Manager or using a published *file mapping* structure. The file mapping is Windows inter-process communication standard supported on all Windows platforms using API functions.

Service running as command line application (`mydualservice TYPE1`). Note that `Ctrl+C` will terminate the service.

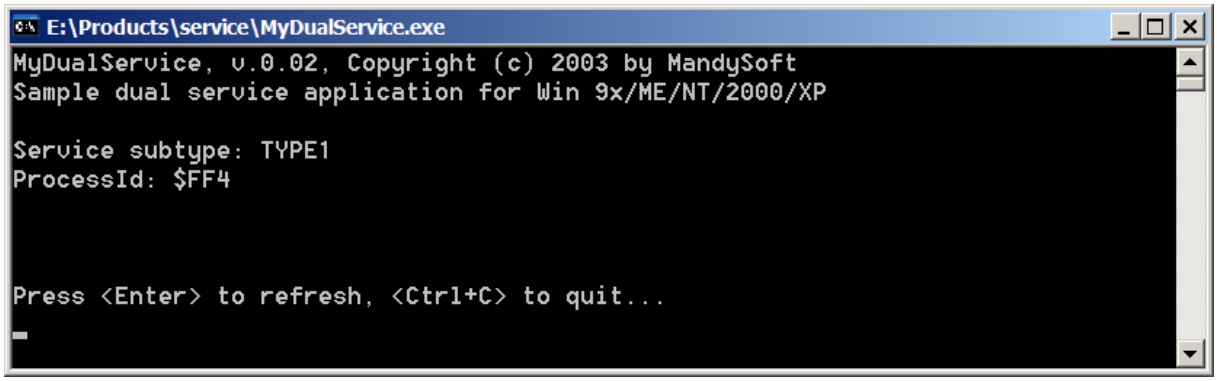


2.3. Status

Using the Service Manager unified interface user can only start, stop, pause or continue a NT service. The file mapping provides in addition also process id, state, custom statistics, etc. The file mapping method can be used from user level via second instance of command line application. The second instance attaches to file mapping record created by running service application (does not matter if running as NT service or command line application) and control its behavior – it can start, stop, pause or continue service similar way as in service manager, it can also display custom statistics. The file mapping record may be mapped to real file using parameter `/F`. This function does work in *Windows 98* but in *Windows XP* it's not possible open such file probably due to insufficient rights or exclusive access.

Such instance can even control service running as NT service. Service state changed using command line application will take effect in service manager (after refreshing using *F5* key).

Second instance asks for service status (`mydualservice TYPE1 /STATUS`). Note there is no custom statistics implemented in demo application.

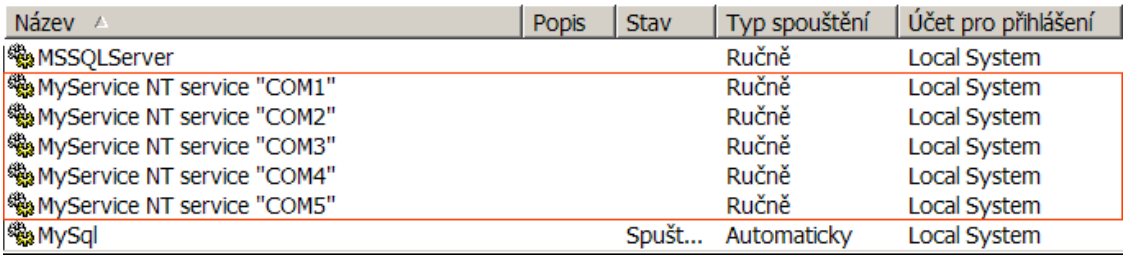


Note that Windows security info may be banned by security manager hence `/SECURITY` switch may not work properly (`Win32 Error: Code: 5`).

2.4. SubType

One service executable can implement more services, each service is identified by *SubType* and can by installed only once to the Service Manager NT service list. So if for example is to be developed service for controlling five GSM modems connected to 5 serial port it's necessary write either a service that in one instance will open 5 serial ports or a service that will as a *SubType* will take port identifier and will be installed 5 times in the Service Manager list.

```
mydualservice COM1 /INSTALL
...
mydualservice COM5 /INSTALL
```



Název	Popis	Stav	Typ spouštění	Účet pro přihlášení
MSSQLServer			Ručně	Local System
MyService NT service "COM1"			Ručně	Local System
MyService NT service "COM2"			Ručně	Local System
MyService NT service "COM3"			Ručně	Local System
MyService NT service "COM4"			Ručně	Local System
MyService NT service "COM5"			Ručně	Local System
MySql		Spušt...	Automaticky	Local System

3. Source code

Complete application source can be found at http://www.2p.cz/download/. Source code is commented thoroughly.

3.1. MyDualService.dpr

```
program MyDualService;
{$APPTYPE CONSOLE}  // it says to Delphi that it's console application (no GUI)
uses
  ComObj,
  Classes,
  SysUtils,
  AuxProj,   // helper for common tasks (INI file, ...)
  CmdLine,   // helper for command line
  Windows,
  { AclApi, { there are errors in Borland's AclApi.H header translation,
              due to static linking to ACLAPI.DLL the DLL is required in system.
              But ACLAPI.DLL is not presented in Windows 9x/ME systems.
              To use ACLAPI.DLL use LoadLibrary function and own function headers }
  AccCtrl,
  Messages,
  SvcMgr,
  WinSvc,
  Math,
  dm_Service in 'dm_Service.pas' {ServiceDM: TDataModule};  // to this module is
encapsulated all service functionality

const
  ServiceName = 'MyService';     { unique service name used for unique objects - file
mapping, service identifier etc. }
```

```
type
{ structure for interprocess communication }
  TMappingRecord = record
    Terminate: Boolean;  // service to be terminated
    Pause: Boolean;      // service to be paused
    Reset: Byte;         // to be reset statistics
    ProcessId: DWORD;    // id of "resident" application
    Status: array[0..1024] of Char;  // human readable service status
  end;

var
  MappingRecord: TMappingRecord;
  FileMapping: THandle;  // handle of shared file mapping
  ServiceStatus: TServiceStatus;  // current service status (paused, running, etc.)
  ServiceStatusHandle: SERVICE_STATUS_HANDLE;
  SvcMachine: PChar;

// if a Windows API returns a error code, RaiseLastWin32Error translates it to an
exception
procedure _RaiseLastWin32Error(aFunc: string);
begin
  try
    RaiseLastWin32Error; // RaiseLastOSError;
  except
    on E: Exception do
    begin
      E.Message:= aFunc+':  '+E.Message;
      raise;
    end;
  end;
end;

// ANSI to OEM conversion, console application uses OEM code page
function CP(const S: string): string;
var
  Buff: array[1..2048] of Char;
begin
  FillChar(Buff, SizeOf(Buff), 0);
  CharToOEMBuff(PChar(S), @Buff, Min(SizeOf(Buff)-1, Length(S)));
  Result:= StrPas(@Buff);
end;

// wrile message to terminal unless running as service that has mandatory /SILENT
command line switch
procedure Write2(const S: string);
var
  D: DWORD;
  S2: string;
  H: THandle;
begin
  if not IsThereCmd('SILENT', clUpcase) then
    Write(CP(S));
  S2:= '';
  if GetCmdString('O:', clUpcase or clValueCase, S2) then
  begin
    if S2<>'' then
    begin
      H:= CreateFile(PChar(S2), GENERIC_READ or GENERIC_WRITE, FILE_SHARE_READ or
FILE_SHARE_WRITE {or FILE_SHARE_DELETE...NTOnly}, nil, OPEN_ALWAYS,
FILE_ATTRIBUTE_NORMAL, 0);
```

```
      try
        if H <> INVALID_HANDLE_VALUE then
        begin
          SetFilePointer(H, 0, nil, FILE_END);
          WriteFile(H, S[1], Length(S), D, nil);
        end;
      finally
        CloseHandle(H);
      end;
    end;
  end;
end;

procedure WriteLn2(const S: string);
begin
  Write2(S+#13#10);
end;

// install/uninstall service to/from list of available services
procedure InstallService(aInstall: Boolean);
var
  SvcMgr, Svc: Integer;
  Path, S1, S2: string;
  P1, P2: PChar;

  function AddPar(aPar: string): string;
  begin
    Result:= '';
    GetCmdString(aPar+':', clUpcase or clValueCase, Result);
    if Result <> '' then
      Result:= ' "/'+aPar+':'+Result+'"';
  end;
resourcestring
  sServiceDisplayName = ServiceName+' NT service "%s"';
begin
  SvcMgr := OpenSCManager(SvcMachine, nil, SC_MANAGER_ALL_ACCESS);
  if SvcMgr = 0 then
    _RaiseLastWin32Error('OpenSCManager');
  try
    if aInstall then
      begin
      // add service entry to service list
      // build path that uses Service Manager to launch service
        Path := ParamStr(0);
        if SvcMachine <> nil then
          Path:= ExpandUNCFileName(Path);

        Path:= Path+' '+SubType+' /SILENT "/I:'+Ini.FileName+'"' + AddPar('F') +
AddPar('O');

        { pass other mandatory/optional parameter here to service using AddPar() ör
append to Path }

        S1:= '';
        GetCmdString('U:', clUpcase or clValueCase, S1);
        if S1 = '' then
          P1:= nil
        else
          P1:= PChar(S1);
        S2:= '';
        GetCmdString('P:', clUpcase or clValueCase, S2);
```

```
        if S2 = '' then
          P2:= nil
        else
          P2:= PChar(S2);
        Svc := CreateService(SvcMgr, PChar(SvcName),
PChar(Format(sServiceDisplayName, [SubType])),
          SERVICE_ALL_ACCESS, SERVICE_WIN32_OWN_PROCESS, SERVICE_DEMAND_START,
SERVICE_ERROR_NORMAL,
          PChar(Path), nil, nil, nil, P1, P2);
        if Svc = 0 then
          _RaiseLastWin32Error('CreateService');
        try
        finally
          CloseServiceHandle(Svc);
        end;
      end
    else
      begin
      // delete service entry from service list
        Svc := OpenService(SvcMgr, PChar(SvcName), SERVICE_ALL_ACCESS);
        if Svc = 0 then
          _RaiseLastWin32Error('OpenService');
        try
          if not DeleteService(Svc) then
            _RaiseLastWin32Error('DeleteService');
        finally
          CloseServiceHandle(Svc);
        end;
      end;
  finally
    CloseServiceHandle(SvcMgr);
  end;
end;

{ test if is running as service (returns True), it's tested by comparing of
  current user  and service user that is defined for service name }
function StartService: Boolean;
var
  Mgr, Svc: Integer;
  UserName, ServiceStartName: string;
  Config: Pointer;
  Size: DWord;
begin
  Result := False;
  Mgr := OpenSCManager(SvcMachine, nil, SC_MANAGER_ALL_ACCESS);
  if Mgr <> 0 then
  begin
    Svc := OpenService(Mgr, PChar(SvcName), SERVICE_ALL_ACCESS);
    Result := Svc <> 0;
    if Result then
    begin
      QueryServiceConfig(Svc, nil, 0, Size);
      Config := AllocMem(Size);
      try
        QueryServiceConfig(Svc, Config, Size, Size);
        ServiceStartName := PQueryServiceConfig(Config)^.lpServiceStartName;
        if CompareText(ServiceStartName, 'LocalSystem') = 0 then
          ServiceStartName := 'SYSTEM';
      finally
        Dispose(Config);
      end;
```

```
      CloseServiceHandle(Svc);
    end;
    CloseServiceHandle(Mgr);
  end;
  if Result then
  begin
    SetLength(UserName, Size);
    GetUserName(PChar(UserName), Size);
    Size := 256;
    SetLength(UserName, StrLen(PChar(UserName)));
    Result := CompareText(UserName, ServiceStartName) = 0;
  end;
end;

{ main service loop for both service and command line application }
procedure DoService;
var
  Msg: TMsg;
  MappingPtr: ^TMappingRecord;
resourcestring
  sStarted = 'Started "%s"';
  sStopped = 'Stopped by remote';
  sTerminating = 'Terminated by itself';
begin
  if InitProc <> nil then
    TProcedure(InitProc);   // not called in Console?, InitComObj of ComObj must be
called!!!

  ServiceDM:= TServiceDM.Create(nil);
  try
    try
      ServiceDM.Log(Format(sStarted, [SubType]));
      MappingPtr:= MapViewOfFile(FileMapping, FILE_MAP_WRITE, 0, 0,
SizeOf(TMappingRecord));
      if MappingPtr = nil then
        _RaiseLastWin32Error('MapViewOfFile');
      try
        MappingPtr^:= MappingRecord;
      finally
        UnmapViewOfFile(MappingPtr);
      end;

      repeat
        // write status and processid as soon as possible, ServiceDM initialization
(e.g. database opening) may take a lot of time
        StrLCopy(@MappingRecord.Status, PChar(ServiceDM.Status),
SizeOf(MappingRecord.Status));
        MappingPtr:= MapViewOfFile(FileMapping, FILE_MAP_WRITE, 0, 0,
SizeOf(TMappingRecord));
        if MappingPtr = nil then
          _RaiseLastWin32Error('MapViewOfFile');
        try
          MappingPtr^.Status:= MappingRecord.Status;
        finally
          UnmapViewOfFile(MappingPtr);
        end;

        Sleep(100);
        MappingPtr:= MapViewOfFile(FileMapping, FILE_MAP_READ, 0, 0,
SizeOf(TMappingRecord));
        if MappingPtr = nil then
```

```
          _RaiseLastWin32Error('MapViewOfFile');
        try
          MappingRecord.Terminate:= MappingPtr^.Terminate;
          { adjust current state if a controlling application changed PAUSE/CONTINUE
state }
          if ServiceDM.Pause <> MappingPtr^.Pause then
          begin
            ServiceDM.Pause:= MappingPtr^.Pause;
            if ServiceDM.Pause then
              ServiceStatus.dwCurrentState:= SERVICE_PAUSED
            else
              ServiceStatus.dwCurrentState:= SERVICE_RUNNING;
          end;
          { adjust current state if service manager forces state change, using
ServiceCtrlHandler }
          if MappingPtr^.Pause and (ServiceStatus.dwCurrentState =
SERVICE_PAUSE_PENDING) then
            ServiceStatus.dwCurrentState:= SERVICE_PAUSED
          else if not MappingPtr^.Pause and (ServiceStatus.dwCurrentState =
SERVICE_CONTINUE_PENDING) then
            ServiceStatus.dwCurrentState:= SERVICE_RUNNING;
          SetServiceStatus (ServiceStatusHandle, ServiceStatus);

          if MappingPtr^.Reset <> MappingRecord.Reset then
          begin
            MappingRecord.Reset:= MappingPtr^.Reset;
            ServiceDM.ResetStatus;
          end;
        finally
          UnmapViewOfFile(MappingPtr);
        end;

        { process message queue }
        if PeekMessage(Msg, 0, 0, 0, PM_REMOVE) then
        begin
          MappingRecord.Terminate:= MappingRecord.Terminate or (Msg.Message =
WM_QUIT);
          if not MappingRecord.Terminate then
          begin
            TranslateMessage(Msg);
            DispatchMessage(Msg);
          end;
        end;
      until MappingRecord.Terminate or ServiceDM.Terminate;
      try
      // close all connected instances here (databases, ports, etc.)
      //  e.g.ServiceDM.Database.Close;
        ServiceDM.LogServiceStatus;
      except
      end;
      if ServiceDM.Terminate then
        ServiceDM.Log(sTerminating)
      else
        ServiceDM.Log(sStopped);
    except
      on E: Exception do
      begin
        ServiceDM.Log(E.Message);
        ExitCode:= 1;
      end;
    end;
```

```
  finally
    ServiceDM.Free;
  end;
end;

{ called by Service Manager to inform runing service that state is changed, e.g. user
manually
  change state in service configuration dialog }
procedure ServiceCtrlHandler(CtrlCode: DWord); stdcall;
var
  MappingPtr: ^TMappingRecord;
begin
  MappingPtr:= MapViewOfFile(FileMapping, FILE_MAP_WRITE, 0, 0,
SizeOf(TMappingRecord));
  if MappingPtr = nil then
    _RaiseLastWin32Error('MapViewOfFile');
  try
    case CtrlCode of
      SERVICE_CONTROL_PAUSE:
        begin
          MappingPtr^.Pause:= True;
          ServiceStatus.dwCurrentState:= SERVICE_PAUSE_PENDING;
        end;
      SERVICE_CONTROL_CONTINUE:
        begin
          MappingPtr^.Pause:= False;
          ServiceStatus.dwCurrentState:= SERVICE_CONTINUE_PENDING;
        end;
      SERVICE_CONTROL_STOP:
        begin
          ServiceStatus.dwWin32ExitCode:= 0;
          ServiceStatus.dwCurrentState:= SERVICE_STOP_PENDING;
          MappingPtr^.Terminate:= True;
        end;
      SERVICE_CONTROL_INTERROGATE:
        begin   // just return service status
        end;
      else
    end;
    SetServiceStatus (ServiceStatusHandle, ServiceStatus);  // do not call
_RaiseLastWin32Error because is called also from console
  finally
    UnmapViewOfFile(MappingPtr);
  end;
end;

{ main procedure executed by service manager, installed by StartServiceCtrlDispatcher
}
procedure ServiceMain(Argc: DWord; Argv: PLPSTR); stdcall;
begin
  FillChar(ServiceStatus, SizeOf(ServiceStatus), 0);
  ServiceStatus.dwServiceType:= SERVICE_WIN32;
  ServiceStatus.dwCurrentState:= SERVICE_START_PENDING;
  ServiceStatus.dwControlsAccepted:= SERVICE_ACCEPT_STOP or
SERVICE_ACCEPT_PAUSE_CONTINUE;

  ServiceStatusHandle:= RegisterServiceCtrlHandler(PChar(SvcName),
@ServiceCtrlHandler);

  if ServiceStatusHandle = 0 then
    _RaiseLastWin32Error('RegisterServiceCtrlHandler');
```

```
// initialization
// Initialization complete - report running status.
  ServiceStatus.dwCurrentState:= SERVICE_RUNNING;
  ServiceStatus.dwCheckPoint:= 0;
  ServiceStatus.dwWaitHint:= 0;

  if not SetServiceStatus (ServiceStatusHandle, ServiceStatus) then
    _RaiseLastWin32Error('SetServiceStatus');

  DoService;
  ServiceStatus.dwCurrentState:= SERVICE_STOPPED;
  SetServiceStatus (ServiceStatusHandle,  ServiceStatus);
end;

procedure WriteSIDInfo(aSID: PSID);
var
  Buff1, Buff2: array[1..512] of Char;
  N1, N2: DWORD;
  Use: SID_NAME_USE;
const
  UseStr: array[sidTypeUser..sidTypeUnknown] of string = ('User', 'Group', 'Domain',
'Alias', 'WellKnownGroup', 'DeletedAccount', 'Invalid', 'Unknown');
resourcestring
  sSid = 'Sid: %s Owner:"%s" Group:"%s"';
begin
  if not LookupAccountSid(SvcMachine, aSid, @Buff1, N1, @Buff2, N2, Use) then
    _RaiseLastWin32Error('LookupAccountSid');
  Writeln2(Format(sSid, [UseStr[Use], @Buff1, @Buff2]));
end;

type
  PPSID = ^PSID;
  PPSECURITY_DESCRIPTOR = ^PSECURITY_DESCRIPTOR;

  SetEntriesInAclProc = function(cCountOfExplicitEntries: ULONG;
pListOfExplicitEntries: PEXPLICIT_ACCESS_;
         OldAcl: PACL; var NewAcl: ACL): DWORD; stdcall;
  GetNamedSecurityInfoProc = function (pObjectName: PAnsiChar; ObjectType:
SE_OBJECT_TYPE;
         SecurityInfo: SECURITY_INFORMATION; ppsidOwner, ppsidGroup: PPSID; ppDacl,
ppSacl: PACL;
         var ppSecurityDescriptor: PSECURITY_DESCRIPTOR): DWORD; stdcall;
  GetSecurityInfoProc = function (handle: THandle; ObjectType: SE_OBJECT_TYPE;
         SecurityInfo: SECURITY_INFORMATION; ppsidOwner, ppsidGroup: PPSID; ppDacl,
ppSacl: PACL;
         var ppSecurityDescriptor: PPSECURITY_DESCRIPTOR): DWORD; stdcall;

var
  S: string;
  H, FileH, LibH: THandle;
  ServiceTableEntry: array[0..1] of TServiceTableEntry;
  MappingPtr: ^TMappingRecord;
  SucurityAttr: TSecurityAttributes;
  SecurityDescriptor: TSecurityDescriptor;
  SecurityDescriptorPtr: PSecurityDescriptor;
//  Access: EXPLICIT_ACCESS;
  Dacl, Sacl: ACL;
  SidOwner, SidGroup: PSID;
  P: Pointer;
  RepeatFlag: Boolean;
```

```
resourcestring
  sAbout1 = 'MyDualService, v.0.02, Copyright (c) 2003 by MandySoft';
  sAbout2 = 'Sample dual service application for Win 9x/ME/NT/2000/XP';
  sHelp1 = 'Usage: MyDualService <type> [/STATUS] [/PAUSE] [/CONTINUE] [/STOP]
[/KILL] [/RESET] [/I:<inifile>] [/H] [/?] [/N:<machine>] [/INSTALL] [/U:<user>]
[/P:<psw>] [/F:<file>] [/UNINSTALL] [/SILENT] [/SECURITY] [/O:<stdout>]';
{ list of available options, such list is not perfect but resource string has limited
length 1024 characters and
  Delphi is unable define resource string array in more handy form }
  sHelp10 = 'Description:';
  sHelp11= '   <type>     connection type';
  sHelp18= '   /PAUSE     pause processing';
  sHelp19= '   /CONTINUE  continue processing';
  sHelp20= '   /STOP      stop process';
  sHelp21= '   /KILL      kill process';
  sHelp22= '   /STATUS    get status';
  sHelp23= '   /RESET     reset status';
  sHelp24= '   /INSTALL   install NT service';
  sHelp25= '   <user>     service account "DomainName\Username", default:
"LocalSystem"';
  sHelp26= '   <psw>      service account password';
  sHelp27= '   <machine>  NT service machine name';
  sHelp28= '   <file>     shared status file';
  sHelp29= '   /UNINSTALL uninstall NT service';
  sHelp30= '   <inifile>  configuration file, default is <name>.INI in program
directory';
  sHelp31= '   /SILENT    no stdout output';
  sHelp32= '   <stdout>   stdout file';
  sHelp33= '   /SECURITY  print security info';
  sHelp34= '   /B         no user input in STATUS';
  sHelp60= 'WARNING:';
  sHelp61= 'Long names containing spaces put as quoted parameter';

  sHelp70= 'Example:';
  sHelp71= '  MyDualService aTYPE';
  sHelp72= '  MyDualService aTYPE /STOP';
  sHelp73= '  MyDualService aTYPE /STATUS';
  sHelp74= '  MyDualService aTYPE /INSTALL /O:stdout.txt /F:aTYPE.1.pid';

  sSubTypeNotSpecified = 'Service subtype not specified or is bad';
  sMasterNotRunning = 'Program not running';
  sErrorCreatingMapping = 'Error creating mapping';
  sProcessId = 'ProcessId: $%x';
  sAlreadyRunning = 'Cannot start, instance already running';
  sInstalling = 'Installing service "%s"';
  sUninstalling = 'Uninstalling service "%s"';
  sPaused = 'PAUSED';
  sSubType = 'Service subtype: %s';
  sCannotLoadLibrary = 'Cannot load library "%s"';
  sPressAnyKey = 'Press <Enter> to quit';
  sPressAnyKey2 = 'Press <Enter> to refresh, <Ctrl+C> to quit...';
begin
  UpdateIni:= False;
  WriteLn2(sAbout1);
  WriteLn2(sAbout2);
  WriteLn2('');
  if IsThereCmd('H', clUpcase) or IsThereCmd('?', clUpcase) then
  begin
    Writeln2(sHelp1);
    WriteLn2('');
    Writeln2(sHelp10);
```

```
    Writeln2(sHelp11);
    Writeln2(sHelp18);
    Writeln2(sHelp19);
    Writeln2(sHelp20);
    Writeln2(sHelp21);
    Writeln2(sHelp22);
    Writeln2(sHelp23);
    Writeln2(sHelp24);
    Writeln2(sHelp25);
    Writeln2(sHelp26);
    Writeln2(sHelp27);
    Writeln2(sHelp28);
    Writeln2(sHelp29);
    Writeln2(sHelp30);
    Writeln2(sHelp31);
    Writeln2(sHelp32);
    Writeln2(sHelp33);
    Writeln2(sHelp34);
    WriteLn2('');
    Writeln2(sHelp60);
    Writeln2(sHelp61);
    WriteLn2('');
    Writeln2(sHelp70);
    Writeln2(sHelp71);
    Writeln2(sHelp72);
    Writeln2(sHelp73);
    Writeln2(sHelp74);
    Exit;
  end;

  repeat
    RepeatFlag:= False;
    try
      S:= '';
      GetCmdString('', 0, S);

      if S = '' then   // check if mandatory type parameter is correct
        raise Exception.Create(sSubTypeNotSpecified);
      { SubType enables implementing more kinds of services in one executable }
      SubType:= S;
      WriteLn2(Format(sSubType, [SubType]));

      S:= '';
      GetCmdString('N:', clUpcase, S);
      if S = '' then
        SvcMachine:= nil
      else
        SvcMachine:= PChar(S);
      SvcName:= ServiceName+'-'+SubType;

      FillChar(MappingRecord, SizeOf(MappingRecord), 0);
      FileH:= INVALID_HANDLE_VALUE;  // = $FFFFFFFF
      FileMapping:= OpenFileMapping(FILE_MAP_WRITE, False,
PChar(ServiceName+'_'+UpperCase(SubType)));
      if (FileMapping = 0) and not (GetLastError() in [ERROR_FILE_NOT_FOUND,
ERROR_INVALID_NAME]) then
        _RaiseLastWin32Error('OpenFileMapping');
      try
        if FileMapping <> 0 then
          begin
          { a service instance already running }
```

```
            CloseHandle(FileMapping);  // open handle have security problem accessing
master's filemapping, MapViewOfFile returns nil
            FileMapping:= CreateFileMapping(INVALID_HANDLE_VALUE, nil,
PAGE_READWRITE, 0, SizeOf(MappingRecord), PChar(ServiceName+'_'+UpperCase(SubType)));
            if IsThereCmd('STATUS', clUpcase) then
              begin
                MappingPtr:= MapViewOfFile(FileMapping, FILE_MAP_READ, 0, 0,
SizeOf(TMappingRecord));
                if MappingPtr = nil then
                  _RaiseLastWin32Error('MapViewOfFile');
                try
                  Writeln2(Format(sProcessId, [MappingPtr^.ProcessId]));
                  Writeln2(MappingPtr^.Status);
                  if MappingPtr^.Pause then
                    Writeln2(sPaused);
                finally
                  UnmapViewOfFile(MappingPtr);
                end;
              end
            else if IsThereCmd('STOP', clUpcase) then
              begin
                ServiceCtrlHandler(SERVICE_CONTROL_STOP);
              end
            else if IsThereCmd('PAUSE', clUpcase) then
              begin
                ServiceCtrlHandler(SERVICE_CONTROL_PAUSE);
              end
            else if IsThereCmd('CONTINUE', clUpcase) then
              begin
                ServiceCtrlHandler(SERVICE_CONTROL_CONTINUE);
              end
            else if IsThereCmd('RESET', clUpcase) then
              begin
                MappingPtr:= MapViewOfFile(FileMapping, FILE_MAP_WRITE, 0, 0,
SizeOf(TMappingRecord));
                if MappingPtr = nil then
                  _RaiseLastWin32Error('MapViewOfFile');
                try
                  Inc(MappingPtr^.Reset);
                finally
                  UnmapViewOfFile(MappingPtr);
                end;
              end
            else if IsThereCmd('SECURITY', clUpcase) then
              begin
                LibH:= LoadLibrary('ADVAPI32.DLL');
                if LibH = 0 then
                  _RaiseLastWin32Error('LoadLibrary');
                try
                  P:= GetProcAddress(LibH, 'GetSecurityInfo');
                  if GetSecurityInfoProc(P)(
                      FileMapping,  //PChar(ServiceName+'_'+UpperCase(SubType)),
                      SE_KERNEL_OBJECT {SE_SERVICE },
                      DACL_SECURITY_INFORMATION or GROUP_SECURITY_INFORMATION or
OWNER_SECURITY_INFORMATION or SACL_SECURITY_INFORMATION,
                      @SidOwner, @SidGroup, @Dacl, @Sacl,
PPSECURITY_DESCRIPTOR(SecurityDescriptorPtr) { it's crazy pointer to pointer as var
param ? }
                    ) <> ERROR_SUCCESS then
                    _RaiseLastWin32Error('GetSecurityInfoProc');
                  try
```

```
                  WriteSIDInfo(SidOwner);
                  WriteSIDInfo(SidGroup);
                finally
                //   LocalFree(SecurityDescriptorPtr); // M according msdn nut
LocalFree need handle
                end;
              finally
                FreeLibrary(LibH);
              end;
            end
          else if IsThereCmd('KILL', clUpcase) then
            begin
              MappingPtr:= MapViewOfFile(FileMapping, FILE_MAP_READ, 0, 0,
SizeOf(TMappingRecord));
              if MappingPtr = nil then
                _RaiseLastWin32Error('MapViewOfFile');
              try
                H:= OpenProcess(PROCESS_ALL_ACCESS, True, MappingPtr^.ProcessId);
                try
                  if not TerminateProcess(H, 0) then
                    _RaiseLastWin32Error('OpenProcess');
                finally
                  CloseHandle(H);
                end;
              finally
                UnmapViewOfFile(MappingPtr);
              end;
            end
          else
            raise Exception.Create(sAlreadyRunning);
        end
      else
        begin
        { no running service instance }
          if IsThereCmd('STATUS', clUpcase) or IsThereCmd('STOP', clUpcase) or
IsThereCmd('KILL', clUpcase) or IsThereCmd('RESET', clUpcase) or IsThereCmd('PAUSE',
clUpcase) or IsThereCmd('CONTINUE', clUpcase) or IsThereCmd('SECURITY', clUpcase)
then
            begin
              raise Exception.Create(sMasterNotRunning);
            end;
          MappingRecord.ProcessId:= GetCurrentProcessId();

          if IsThereCmd('INSTALL', clUpcase) then
            begin
              Writeln2(Format(sInstalling, [SvcName]));
              InstallService(True);
            end
          else if IsThereCmd('UNINSTALL', clUpcase) then
            begin
              Writeln2(Format(sUninstalling, [SvcName]));
              InstallService(False)
            end
          else
            begin
              SucurityAttr.nLength:= SizeOf(SucurityAttr);
              SucurityAttr.bInheritHandle:= True;
              SucurityAttr.lpSecurityDescriptor:= @SecurityDescriptor;
              InitializeSecurityDescriptor(SucurityAttr.lpSecurityDescriptor,
SECURITY_DESCRIPTOR_REVISION);
  (*
```

```
                SetSecurityDescriptorControl(SucurityAttr.lpSecurityDescriptor,
SE_DACL_AUTO_INHERIT_REQ or SE_SACL_AUTO_INHERIT_REQ, SE_DACL_AUTO_INHERIT_REQ or
SE_SACL_AUTO_INHERIT_REQ);
                InitializeAcl(Dacl, SizeOf(Dacl), 1{ACL_REVISION});

                Access.grfAccessPermissions:= SECTION_ALL_ACCESS;
                Access.grfAccessMode:= GRANT_ACCESS;
                Access.grfInheritance:= NO_INHERITANCE;
                Access.Trustee.MultipleTrusteeOperation:= NO_MULTIPLE_TRUSTEE;
                // change these informations to grant access to a group or other user
                Access.Trustee.TrusteeForm:= TRUSTEE_IS_NAME;
                Access.Trustee.TrusteeType:= TRUSTEE_IS_USER;
                Access.Trustee.ptstrName:= 'CURRENT_USER';  // 'CREATOR OWNER'

                LibH:= LoadLibrary('ADVAPI32.DLL');
                if LibH = 0 then
                  _RaiseLastWin32Error('LoadLibrary');
                try
                  P:= GetProcAddress(LibH, 'SetEntriesInAclA');
                  if P <> nil then
                    SetEntriesInAclProc(P)(1, @Access, nil, Dacl);
                finally
                  FreeLibrary(LibH);
                end;
  *)
                SetSecurityDescriptorDacl(SucurityAttr.lpSecurityDescriptor, True,
nil {@Dacl{ nil = all access }, False);

                S:= '';
                GetCmdString('F:', clUpcase or clValueCase, S);
                if S <> '' then
                begin
                  FileH:= CreateFile(PChar(S), GENERIC_WRITE or GENERIC_READ,
FILE_SHARE_READ or FILE_SHARE_WRITE{?}, @SucurityAttr, CREATE_ALWAYS,
FILE_ATTRIBUTE_TEMPORARY or FILE_FLAG_DELETE_ON_CLOSE, 0);
                  { file is inaccessable in WinXP, not enough rights, even after
mapping is closed, probably created file is impossible to share }
                  if FileH = INVALID_HANDLE_VALUE then
                    _RaiseLastWin32Error('CreateFile');
                end;
                FileMapping:= CreateFileMapping(FileH, @SucurityAttr, PAGE_READWRITE,
0, SizeOf(MappingRecord), PChar(ServiceName+'_'+UpperCase(SubType)));
                if FileMapping = 0 then
                  _RaiseLastWin32Error('CreateFileMapping');
                if StartService then
                  begin
                  { called by service manager to start service }
                    FillChar(ServiceTableEntry, SizeOf(ServiceTableEntry), 0);
                    ServiceTableEntry[0].lpServiceName:= PChar(SvcName);
                    ServiceTableEntry[0].lpServiceProc:= @ServiceMain;
                    if not StartServiceCtrlDispatcher(ServiceTableEntry[0]) then
                      _RaiseLastWin32Error('StartServiceCtrlDispatcher');
                  end
                else  { called from command line }
                  DoService;
              end;
          end;
      finally
        if FileMapping <> 0 then
          CloseHandle(FileMapping);
        if FileH <> INVALID_HANDLE_VALUE then
```

```
          CloseHandle(FileH);
      end;
      ExitCode:= 0;
    except
      on E: Exception do
      begin
        Writeln2(E.Message);
        ExitCode:= 1;
        if not IsThereCmd('SILENT', clUpcase) and not IsThereCmd('B', clUpcase) and
not IsThereCmd('STATUS', clUpcase) then
        begin
        { stop output listing not to disappear if executed from GUI }
          WriteLn2('');
          WriteLn2(sPressAnyKey);
          ReadLn;
        end;
      end;
    end;
    if not IsThereCmd('SILENT', clUpcase) and not IsThereCmd('B', clUpcase) then
    begin
    { stop output listing not to disappear if executed from GUI and enable STATUS
refresh without reexecution }
      WriteLn2('');
      if IsThereCmd('STATUS', clUpcase) then
      begin
        WriteLn2(sPressAnyKey2);
        RepeatFlag:= True;
        ReadLn;
      end
    end;
  until not RepeatFlag;
end.
```

3.2. dm_Service.pas

```
unit dm_Service;

interface

uses
  Windows, Messages, SysUtils, Classes, Controls, Forms, Connect;

type
  TServiceDM = class(TDataModule)
    Logger: TFileLogger;
    procedure DataModuleCreate(Sender: TObject);
  private
  public
    Terminate: Boolean;
    Pause: Boolean;
    procedure Log(S: string);
    function Status: string;
    procedure ResetStatus;
    procedure LogServiceStatus;
  end;

var
  ServiceDM: TServiceDM;

  SubType: string;
```

```
  SvcName: string;

implementation
uses
  AuxProj;

{$R *.DFM}

procedure TServiceDM.DataModuleCreate(Sender: TObject);
begin
//  Logger.LogFile:= Ini.ReadString('Global', 'LogFile', 'NUL');  // assign a log
file to logger
end;

procedure TServiceDM.Log(S: string);
begin
  try
    Logger.Log('', lchNull, S);
  except
  end;
end;

function TServiceDM.Status: string;
begin
  Result:= '';
// write status, custom statistics etc., e.g. number of running time, received
messages, procesed records, ...
  Result:= Result+#13#10;
end;

procedure TServiceDM.ResetStatus;
begin
// reset custom statistics, e.g. number of received messages, processed records, ...
end;

procedure TServiceDM.LogServiceStatus;
begin
// if necessary log service status for external application (watch dog, etc.)
end;

end.
```