```
unit ProtConst;

interface

Uses

  SysUtils;

type

  TDeviceId = ( diPC, diNM);
  ByteArray = Array Of Byte;

Var

  pcMsgFilter : Set Of Byte;
  pcChFilter  : Set Of Byte;

// Exported functions

  Function SectionName( aSectionId: Byte): String;
  Function SlotIdName ( aSlotId   : Byte): String;
  Function Crack      ( aDeviceId: TDeviceId; aSize: Cardinal; aData: ByteArray; Out aRaw, anOctal: String): String;

Type

    TSetSynthSettings = Procedure(
      RecType          : Byte;
      InternalClock    : Byte;
      MinVelocityScale : Byte;
      MaxVelocityScale : Byte;
      LedsActive       : Byte;
      Tempo            : Byte;
      LocalOff         : Byte;
      KeyboardMode     : Byte;
      PedalPolarity    : Byte;
      GlobalSync       : Byte;
      MasterTune       : SmallInt;
      ProgChangeRx     : Byte;
      ProgChangeTx     : Byte;
      KnobMode         : Byte;
      SynthName        : String;
      MidiActiveA      : Byte;
      MidiChannelA     : Byte;
      MidiActiveB      : Byte;
      MidiChannelB     : Byte;
      MidiActiveC      : Byte;
      MidiChannelC     : Byte;
      MidiActiveD      : Byte;
      MidiChannelD     : Byte
    ) Of Object;

    TSetPatchSettings = Procedure(
      Slot               : Byte;
      KeyRangeMin        : Byte;
      KeyRangeMax        : Byte;
      VelocityRangeMin   : Byte;
      VelocityRangemax   : Byte;
      BendRange          : Byte;
      PortamentoTime     : Byte;
      PortamentoMode     : Byte;
      RequestedVoices    : Byte;
      PedalMode          : Byte;
      OctaveShift        : Byte;
      VoiceRetrigPoly    : Byte;
      VoiceRetrigCommon  : Byte;
      DividerBarPosition : Word;
      RedVisible         : Byte;
      BlueVisible        : Byte;
      YellowVisible      : Byte;
      GrayVisible        : Byte;
      GreenVisible       : Byte;
      PurpleVisible      : Byte;
      WhiteVisible       : Byte
    ) Of Object;

    TVoicesGranted = Procedure(
      ACount,
      BCount,
      CCount,
      DCount:
      Byte) Of Object;

    TSetParameter = Procedure(
      Slot      : Byte;
      Section   : Byte;
      Module    : Byte;
      Parameter : Byte;
      Value     : Byte
    ) Of Object;

    TSetActiveChannel = Procedure( aValue: Byte) Of Object;
    TSetChannelmask   = Procedure( aValue: Byte) Of Object;

Const
```

```
  OnSetSynthSettings : TSetSynthSettings = Nil;
  OnSetPatchSettings : TSetPatchSettings = Nil;
  OnVoicesGranted    : TVoicesGranted    = Nil;
  OnSetParameter     : TSetParameter     = Nil;
  OnSetActiveChannel : TSetActiveChannel = Nil;
  OnSetChannelMask   : TSetChannelmask   = Nil;


implementation


  Procedure Grow( Var S: ByteArray; anAmount: Integer = 1);
  Begin
    SetLength( S, Length( S) + anAmount);
  End;

  Procedure Append( Var S: ByteArray; aByte: Byte);
  Begin
    Grow( S);
    S[ Length( S) - 1] := aByte;
  End;

  Procedure AddIn( Var S: ByteArray; aByte: Byte);
  Begin
    S[ Length( S) - 1] := S[ Length( S) - 1] + aByte;
  End;

  Procedure ShiftInSeptet( aSeptet: Byte; Var S: ByteArray; Var aBitIndex: Byte);
  Begin
    // Now maybe modify the last byte of S, and maybe add a byte as well.
    Case aBitIndex Of
      0 :
        // every 1st time we must not add in, but append only
        {$R-}
        Append( S, aSeptet Shl 1);
        {$R+}
      7 :
        // Every 8th time aSeptet fits in exactly in the last byte of S.
        AddIn( S, aSeptet);
      Else Begin
        // All other times not only the last char of S must be modified
        // by a part of the septet but also a byte must be added containing
        // the 'leftovers' of the septet.
        AddIn( S, aSeptet Shr( 7 - aBitIndex));
        {$R-}
        Append( S, aSeptet Shl ( aBitIndex + 1));
        {$R+}
      End;
    End;
    aBitIndex := ( aBitIndex + 8 - 7) Mod 8;
  End;

  Procedure AppendSeptet( Var S: ByteArray; aSeptet: Byte; Var aBitIndex: Byte);
  Begin
    Append( S, aSeptet And $7f);
  End;

  Procedure ShiftInOctet( anOctet: Byte; Var S: ByteArray; Var aBitIndex: Byte);
  Begin
    Case aBitIndex Of
      0 : Append( S, anOctet);
      Else Begin
        // All other times not only the last char of S must be modified
        // by a part of the octet but also a byte must be added containing
        // the 'leftovers' of the octet.
        AddIn( S, anOctet Shr ( 8 - aBitIndex));
        {$R-}
        Append( S, anOctet Shl aBitIndex);
        {$R+}
      End;
    End;
    // aBitIndex is not changed : aBitIndex := ( aBitIndex + 8 - 8) Mod 8;
  End;

  Procedure AppendOctet( Var S: ByteArray; anOctet: Byte; Var aBitIndex: Byte);
  Begin
    // aBitIndex is not changed
    Append( S, anOctet);
  End;

  Procedure AddBitStuff( Const aFrom: ByteArray; Var aTo: ByteArray; Var aBitIndex: Byte; aStartIndex: Integer);
  Var
    i : Integer;
  Begin
    For i := aStartIndex To Length( aFrom) - 2 Do // Don't add checksum
      ShiftInSeptet( aFrom[ i], aTo, aBitIndex);      // Add next septet
  End;

  Procedure BitStuff( Const aFrom: ByteArray; Var aTo: ByteArray; aStartIndex: Integer);
  Var
    aBitIndex : Byte;
  Begin
    Finalize( aTo);  // Clear result
    aBitIndex := 0;  // No septets added sofar
```

```
  AddBitStuff( aFrom, aTo, aBitIndex, aStartIndex);
End;

Function HexDump( aData: ByteArray; aStartIndex: Integer = 0; anEndIndex: Integer = MaxInt): String; // Returns
aData in readable format
Var
  i : Integer;
Begin
  Result := '';
  If anEndIndex > Length( aData) - 1
  Then anEndIndex := Length( aData) - 1;
  For i := aStartIndex To anEndIndex Do
    Result := Result + Format( '%.2x ', [ aData[ i]]);
End;

Function SectionName( aSectionId: Byte): String;
Begin
  Case aSectionId Of
    0 : Result := 'global';
    1 : Result := 'poly';
    2 : Result := 'morph';
  Else Result := Format( 'section (#%d) ?', [ aSectionId])
  End;
End;

Function SlotIdName( aSlotId: Byte): String;
Begin
  Case aSlotId Of
    0 : Result := 'A';
    1 : Result := 'B';
    2 : Result := 'C';
    3 : Result := 'D';
  Else
    Result := Format( 'invalid slot id (%d) ?', [ aSlotId]);
  End;
End;

Function SlotmaskName( aSlots: Byte): String;
Begin
  Result := '';
  If aSlots And $08 = 0
  Then Result := Result + '-'
  Else Result := Result + 'A';
  If aSlots And $04 = 0
  Then Result := Result + '-'
  Else Result := Result + 'B';
  If aSlots And $02 = 0
  Then Result := Result + '-'
  Else Result := Result + 'C';
  If aSlots And $01 = 0
  Then Result := Result + '-'
  Else Result := Result + 'D';
End;

Function KnobName( aKnob: Byte): String;
Begin
  Case aKnob Of
    0 .. 17 : Result := Format( 'kn %.2d', [ aKnob + 1]);
    19      : Result := 'pedal';
    20      : Result := 'after';
    22      : Result := 'on/of';
    Else      Result := Format( 'invalid knob name (%d) ?', [ aKnob])
  End;
End;

Function OnOffName( aValue : Byte): String;
Begin
  Case aValue Of
    $00 : Result := 'Off';
    $01 : Result := 'On ';
    Else Result := Format( 'Undefined on/off (%d)', [ aValue]);
  End;
End;


Function Crack( aDeviceId: TDeviceId; aSize: Cardinal; aData: ByteArray; Out aRaw, anOctal: String): String;
Var
  OctetData : ByteArray;


  Function ChkStr( Applicable: Boolean): String; // Returns the checksum in readable format
  Var
    ChkSum : Integer;
    i      : Integer;
  Begin
    If Applicable
    Then Begin
      ChkSum := ( $f0 + $33) And $7f;
      For i := 0 To aSize - 2 Do
        ChkSum := ( ChkSum + aData[ i]) And $7f;
      If ChkSum = aData[ aSize - 1]
      Then Result := 'ok      ' // checksum ok --> --> --> --> --> --> --> --\
      Else Result := Format( '[%.2x-%.2x] ', [ ChkSum, aData[ aSize - 1]]); // not ok
    End
    Else Result := '-       '; // not present
```

```
End;

Function PidStr( Applicable: Boolean): String; // PatchId - readable
Begin
  If Applicable
  Then Result := Format( '%.2x ', [ aData[ 2]])
  Else Result := '-  ';
End;

Function SlotStr( Applicable: Boolean): String; // Slot Id - readable
Begin
  If Applicable
  Then Result := SlotIdName( aData[ 0] And $03) + ' '
  Else Result := '- ';
End;

Function CrackSynthSettings( anExtra: Boolean): String;
Var
  i : Integer;
Var
  RecType          : Byte;
  InternalClock    : Byte;
  MinVelocityScale : Byte;
  MaxVelocityScale : Byte;
  LedsActive       : Byte;
  Tempo            : Byte;
  LocalOff         : Byte;
  KeyboardMode     : Byte;
  PedalPolarity    : Byte;
  GlobalSync       : Byte;
  MasterTune       : ShortInt;
  ProgChangeRx     : Byte;
  ProgChangeTx     : Byte;
  KnobMode         : Byte;
  SynthName        : String;
  MidiActiveA      : Byte;
  MidiChannelA     : Byte;
  MidiActiveB      : Byte;
  MidiChannelB     : Byte;
  MidiActiveC      : Byte;
  MidiChannelC     : Byte;
  MidiActiveD      : Byte;
  MidiChannelD     : Byte;
Begin
  BitStuff( aData, OctetData, 3);
  {$R-}
  RecType          :=    OctetData[ 0]; // ?? 03 why ??
  InternalClock    :=  ( OctetData[ 1] Shr 7) And $01;
  MinVelocityScale := (( OctetData[ 1]      ) And $7f);
  LedsActive       :=  ( OctetData[ 2] Shr 7) And $01;
  MaxVelocityScale := (( OctetData[ 2]      ) And $7f);
  Tempo            := (( OctetData[ 3]      ) And $ff);
  LocalOff         :=  ( OctetData[ 4] Shr 7) And $01;
  KeyboardMode     :=  ( OctetData[ 4] Shr 6) And $01;
  PedalPolarity    :=  ( OctetData[ 4] Shr 5) And $01;
  GlobalSync       :=  ( OctetData[ 4]      ) And $1f;
  MasterTune       := (( OctetData[ 5]      ) And $ff);
  ProgChangeRx     :=  ( OctetData[ 6] Shr 7) And $01;
  ProgChangeTx     :=  ( OctetData[ 6] Shr 6) And $01;
  KnobMode         :=  ( OctetData[ 6] Shr 5) And $01;
  {$R+}
  SynthName        := '';
  i := 7;
  While ( OctetData[ i] <> 0) And ( Length( SynthName) < 16) Do
  Begin
    SynthName := SynthName + Char( OctetData[ i]);
    Inc( i);
  End;
  If Length( SynthName) < 16
  Then Inc( i); // Skip \0 if present.
  MidiChannelA     :=  ( Byte   ( OctetData[ i]) Shr 0) And $0f;
  MidiActiveA      :=  ( Byte   ( OctetData[ i]) Shr 4) And $01; Inc( i, 2);
  MidiChannelB     :=  ( Byte   ( OctetData[ i]) Shr 0) And $0f;
  MidiActiveB      :=  ( Byte   ( OctetData[ i]) Shr 4) And $01; Inc( i, 2);
  MidiChannelC     :=  ( Byte   ( OctetData[ i]) Shr 0) And $0f;
  MidiActiveC      :=  ( Byte   ( OctetData[ i]) Shr 4) And $01; Inc( i, 2);
  MidiChannelD     :=  ( Byte   ( OctetData[ i]) Shr 0) And $0f; Inc( i, 2);
  MidiActiveD      :=  ( Byte   ( OctetData[ i]) Shr 4) And $01;
  If Assigned( OnSetSynthSettings)
  Then
    OnSetSynthSettings(
      RecType          ,
      InternalClock    ,
      MinVelocityScale ,
      MaxVelocityScale ,
      LedsActive       ,
      Tempo            ,
      LocalOff         ,
      KeyboardMode     ,
      PedalPolarity    ,
      GlobalSync       ,
      MasterTune       ,
      ProgChangeRx     ,
      ProgChangeTx     ,
      KnobMode         ,
```

```
        SynthName       ,
        MidiActiveA     ,
        MidiChannelA    ,
        MidiActiveB     ,
        MidiChannelB    ,
        MidiActiveC     ,
        MidiChannelC    ,
        MidiActiveD     ,
        MidiChannelD
      );

    Result := '';
    If anExtra
    Then Begin
      If OctetData[ i] = $07
      Then Begin
        Inc( i);
        Result := Result + Format( 'slots_active(%s) ', [ SlotMaskName( OctetData[ i])]);
        Inc( i);
        If OctetData[ i] = $09
        Then Begin
          Inc( i);
          Result := Result + Format( 'active_slot(%s) ', [ SlotIdName( OctetData[ i])]);
          Inc( i);
          If OctetData[ i] = $05
          Then Begin
            Inc( i);
            Result := Result + Format( 'voice_counts(#%d,#%d,#%d,#%d) ', [ OctetData[ i], OctetData[ i + 1],
OctetData[ i + 2], OctetData[ i + 3]]);
            If Assigned( OnVoicesGranted)
            Then OnVoicesGranted( OctetData[ i], OctetData[ i + 1], OctetData[ i + 2], OctetData[ i + 3]);
            // Inc( i, 4);
          End
          Else Result := Result + '[05 expected] ';
        End
        Else Result := Result + '[09 expected] ';
      End
      Else Result := Result + '[07 expected] ';
    End;
    Result := Result + HexDump( OctetData);
  End;

  Function CrackPatchSettings: String;
  Var
    Slot               : Byte;
    KeyRangeMin        : Byte;
    KeyRangeMax        : Byte;
    VelocityRangeMin   : Byte;
    VelocityRangemax   : Byte;
    BendRange          : Byte;
    PortamentoTime     : Byte;
    PortamentoMode     : Byte;
    RequestedVoices    : Byte;
    PedalMode          : Byte;
    OctaveShift        : Byte;
    VoiceRetrigPoly    : Byte;
    VoiceRetrigCommon  : Byte;
    DividerBarPosition : Word;
    RedVisible         : Byte;
    BlueVisible        : Byte;
    YellowVisible      : Byte;
    GrayVisible        : Byte;
    GreenVisible       : Byte;
    PurpleVisible      : Byte;
    WhiteVisible       : Byte;
  Begin
    {$R-}
    Slot               :=    aData[  0] And $03;
    KeyRangeMin        := (( aData[  4] And $3f) Shl 1) + (( aData[  5] And $40) Shr 6);
    KeyRangeMax        := (( aData[  5] And $3f) Shl 1) + (( aData[  6] And $40) Shr 6);
    VelocityRangeMin   := (( aData[  6] And $3f) Shl 1) + (( aData[  7] And $40) Shr 6);
    VelocityRangemax   := (( aData[  7] And $3f) Shl 1) + (( aData[  8] And $40) Shr 6);
    BendRange          :=  ( aData[  8] And $3e) Shr 1;
    PortamentoTime     := (( aData[  8] And $01) Shl 6) + (( aData[  9] And $7e) Shr 1);
    PortamentoMode     :=    aData[  9] And $01;
    RequestedVoices    :=  ( aData[ 10] And $3e) Shr 1;
    PedalMode          :=  ( aData[ 10] And $40) Shr 6;
    DividerBarPosition :=  ( Word( aData[ 11] And $4f)  Shl 6) + ( Word( aData[ 12] And $7e) Shr 1);
    OctaveShift        := (( aData[ 12] And $01) Shl 2) + (( aData[ 13] And $60) Shr 5);
    VoiceRetrigPoly    :=  ( aData[ 14] And $08) Shr 3;
    VoiceRetrigCommon  :=  ( aData[ 14] And $10) Shr 4;
    GreenVisible       :=  ( aData[ 13] And $01) Shr 0;
    GrayVisible        :=  ( aData[ 13] And $02) Shr 1;
    YellowVisible      :=  ( aData[ 13] And $04) Shr 2;
    BlueVisible        :=  ( aData[ 13] And $08) Shr 3;
    RedVisible         :=  ( aData[ 13] And $10) Shr 4;
    WhiteVisible       :=  ( aData[ 14] And $20) Shr 5;
    PurpleVisible      :=  ( aData[ 14] And $40) Shr 6;
    {$R+}
    If Assigned( OnSetPatchSettings)
    Then
      OnSetPatchSettings(
        Slot            ,
        KeyRangeMin     ,
        KeyRangeMax     ,
```

```
        VelocityRangeMin  ,
        VelocityRangemax  ,
        BendRange         ,
        PortamentoTime    ,
        PortamentoMode    ,
        RequestedVoices   ,
        PedalMode         ,
        OctaveShift       ,
        VoiceRetrigPoly   ,
        VoiceRetrigCommon ,
        DividerBarPosition,
        RedVisible        ,
        BlueVisible       ,
        YellowVisible     ,
        GrayVisible       ,
        GreenVisible      ,
        PurpleVisible     ,
        WhiteVisible
      );
    Result := Format( 'See tabsheet ''Slot %s''', [ SlotIdName( Slot)])
  End;

  Procedure SetOctalOutput( aStartIndex: Integer);
  Begin
    BitStuff( aData, OctetData, aStartIndex);
    anOctal := HexDump( OctetData);
  End;

Var
  anOpCode  : Byte;
  aSubCode  : Byte;
  aSubSub   : Byte;
  aPid      : Byte;
  HasCheck  : Boolean;
  HasPid    : Boolean;
  IsSlotMsg : Boolean;
Begin
  Result  := '';
  aRaw    := '';
  anOctal := '';
  If aSize < 3
  Then Exit;

  HasCheck  := False;
  HasPid    := False;
  IsSlotMsg := False;
  anOpCode  := aData[ 0] Shr 2;

  If anOpCode In pcMsgFilter
  Then Exit;

  Case anOpCode Of
    $00 : Begin // 00 : I Am (no slot info, no check, no pid)
        Result := 'I Am ';
        aSubCode := aData[ 2];
        Case aSubCode Of
          $00 : Result := Result + Format( 'req   vers = %d.%.2d', [ aData[ 3], aData[ 4]]);
          $01 : Result := Result + Format( 'reply vers = %d.%.2d', [ aData[ 3], aData[ 4]]);
          Else  Result := Result + Format( 'unknown sub command %.2x', [ aSubCode]);
        End;
      End;
    $13 : Begin // 4c .. 4f : Param (no check)
        HasPid    := True;
        IsSlotMsg := True;
        Result    := 'Param ';
        aSubCode  := aData[ 3];
        Case aSubCode Of
          $2f : Result := Result + Format( 'select (dir??=#%d,sect=%s,mod=#%d,parm=#%d)', [ aData[ 4], SectionName(
aData[ 5]), aData[ 6], aData[ 7]]);
          $40 : Begin
              Result :=
                Result +
                Format(
                  'change (sect=%s,mod=#%d,parm=#%d) to #%d',
                  [
                    SectionName( aData[ 4]),
                    aData[ 5],
                    aData[ 6],
                    aData[ 7]
                  ]
                );
              If Assigned( OnSetParameter)
              Then
                OnSetParameter(
                  aData[ 0] And $03,
                  aData[ 4],
                  aData[ 5],
                  aData[ 6],
                  aData[ 7],
                );
            End;
          $43 : Result := Result + Format( 'morph change (sect=%s,mod=#%d,parm=#%d) to #%d : dir=#%d', [
SectionName( aData[ 4]), aData[ 5], aData[ 6], aData[ 7], aData[ 8]]);
          Else  Result := Result + Format( 'unknown sub command %.2x', [ aSubCode]);
        End;
```

```
      End;
    $14 : Begin // 50 .. 53 : NmInfo
        HasPid    := True;
        HasCheck  := True;
        IsSlotMsg := True;
        Result    := 'NM info ';
        aSubCode  := aData[ 3];
        Case aSubCode Of
          $05 : Begin
              IsSlotMsg := False;
              Result := Result + Format( 'voice counts (#%d,#%d,#%d,#%d)', [ aData[ 4], aData[ 5], aData[ 6], aData[
7]]);
              If Assigned( OnVoicesGranted)
              Then OnVoicesGranted( aData[ 4], aData[ 5], aData[ 6], aData[ 7]);
            End;
          $07 : Begin
              IsSlotMsg := False;
              Result := Result + Format( 'Slot selection (%s)', [ Slotmaskname( aData[ 4])]);
            End;
          $09 : Begin
              IsSlotMsg := False;
              Result := Result + Format( 'Active slot (%s)', [ SlotIdName( aData[ 4])]);
            End;
          $39 : Result := Result + 'lights - 8 bytes';
          $3a : Result := Result + 'VU / SEQ change 1 + 10 bytes';
          $40 : Result := Result + Format( 'hw knob change (%s,mod=#%d,parm=#%d,val=#%d)', [ SectionName( aData[
4]), aData[ 5], aData[ 6], aData[ 7]])
          Else  Result := Result + Format( 'unknown sub command %.2x', [ aSubCode]);
        End;
      End;
    $16 : Begin // 58 .. 5b : Ack
        HasPid    := True;
        HasCheck  := True;
        IsSlotMsg := True;
        Result    := 'Ack, or data ..';
      End;
    $17 : Begin // 5c .. 5f :
        aPid      := aData[ 2];
        HasPid    := aPid < $40;
        HasCheck  := True;
        IsSlotMsg := True;
        If HasPid
        Then Begin
          aSubCode  := aData[ 3];
          Case aSubCode Of
            $20 : Result := Result + 'get patch settings';
            $25 : Result := Result + 'knob assignment    ' + KnobName( aData[ 6]);
            $26 : Result := Result + 'knob assign change ' + KnobName( aData[ 4]);
            $32 : Result := Result + Format( 'module deletions (%s,#%d)', [ SectionName( aData[ 4]), aData[ 5]]);
            $34 : Result := Result + Format( 'module moves (%s,mod=#%d,x=#%d,y=#%d)', [ SectionName( aData[ 4]),
aData[ 5], aData[ 6], aData[ 7]]);
            $4b : Result := Result + Format( 'Get section info 1 (%s) ??', [ SectionName( aData[ 4])]);
            $4f : Result := Result + Format( 'Get section info 2 (%s) ??', [ SectionName( aData[ 4])]);
            $50 : Result := Result + 'cable insertion';
            $51 : Result := Result + 'cable deletion';
            $53 : Result := Result + Format( 'Get section info 3 (%s) ??', [ SectionName( aData[ 4])]);
            $54 : Result := Result + 'cable move';
            $55 : Begin
                IsSlotMsg := False;
                Result := Result + 'Send controller snapshot';
              End;
            $56 : Result := Result + Format( 'Send note (%s,#%d)', [ OnOffName( aData[ 4]), aData[ 5]]);
            $68 : Result := Result + 'Get current notes';
            $70 : Begin
                IsSlotMsg := False;
                Result := Result + 'Upload active slot (to PC)';
              End;
            Else  Result := Result + Format( '[17 PID] unknown sub command %.2x', [ aSubCode])
          End;
        End
        Else Begin
          aSubCode  := aData[ 2];
          Case aSubCode Of
            $41 : Begin
                Result  := Result + 'pm '; // patch manager - but not really
                aSubSub := aData[ 3];
                Case aSubSub Of
                  $07 : Begin
                      Result := Result + Format( 'slot change (%s)', [ SlotMaskName( aData[ 4])]);
                      If Assigned( OnSetChannelmask)
                      Then OnSetChannelMask( aData[ 4]);
                    End;
                  $09 : Begin
                      IsSlotMsg := False;
                      Result := Result + Format( 'set active slot (%s)', [ SlotIdName( aData[ 4])]);
                      If Assigned( OnSetActiveChannel)
                      Then OnSetActiveChannel( aData[ 4]);
                    End;
                  $0a : Result := Result + Format( 'request patch (#%d,#%d,#%d)', [ aData[ 4], aData[ 5], aData[
6]]);
                  $14 : Begin
                      IsSlotMsg := False;
                      Result := Result + Format( 'get patch list from (#%d)', [ Integer( aData[ 5]) + 128 * Integer(
aData[ 4])]);
                    End;
```

```
              else  Result := Result + Format( '[17 41] unknown subsub %.2x', [ aSubSub]);
            End;
          End;
        $44 : Result := 'Get ext. synth settings';
        Else Result := Result + Format( '[17] unknown sub command %.2x', [ aSubCode]);
      End;
    End;
  End;
$1c : Begin // 70 .. 73
    aPid      := aData[ 2];
    HasPid    := aPid < $40;
    HasCheck  := True;
    IsSlotMsg := True;
    If HasPid
    Then Begin
      aSubCode  := aData[ 3];
      Result := Result + Format( '[1c PID] unknown sub command %.2x', [ aSubCode]);
    End
    Else Begin
      aSubCode := aPid;
      Result := Result + Format( '[1c] unknown sub command %.2x', [ aSubCode]);
      SetOctalOutput( 3);
    End;
  End;
$1d : Begin // 74 .. 77
    aPid      := aData[ 2];
    HasPid    := aPid < $40;
    HasCheck  := True;
    IsSlotMsg := True;
    If HasPid
    Then Begin
      aSubCode  := aData[ 3];
      Result := Result + Format( '[1d PID] unknown sub command %.2x', [ aSubCode]);
    End
    Else Begin
      aSubCode := aPid;
      Result := Result + Format( '[1d] unknown sub command %.2x', [ aSubCode]);
      SetOctalOutput( 3);
    End;
  End;
$1e : Begin // 78 .. 7b
    aPid      := aData[ 2];
    HasPid    := aPid < $40;
    HasCheck  := True;
    IsSlotMsg := True;
    If HasPid
    Then Begin
      aSubCode  := aData[ 3];
      Case aSubCode Of
        $01 : Result := '    ext. synth settings ' + CrackSynthSettings( True);
        $10 : Result := '    patch settings '          + CrackPatchSettings;
        $25 : Result := Format( '    section info 1 (%s) ??', [ SectionName(( aData[ 4] And $20) Shr 5)]);
        $29 : Result := Format( '    section info 3 (%s) ??', [ SectionName(( aData[ 4] And $20) Shr 5)]);
        $2d : Result := Format( '    section info 2 (%s) ??', [ SectionName(( aData[ 4] And $20) Shr 5)]);
        $34 : Result := '    current notes';
        Else Result := Result + Format( '[1e PID] unknown sub command %.2x', [ aSubCode]);
      End;
    End
    Else Begin
      aSubCode := aPid;
      Result := Result + Format( '[1e] unknown sub command %.2x', [ aSubCode]);
      SetOctalOutput( 3);
    End;
  End;
$1f : Begin // 7c .. 7f
    aPid      := aData[ 2];
    HasPid    := aPid < $40;
    HasCheck  := True;
    IsSlotMsg := True;
    If hasPid
    Then Begin
      aSubCode  := aData[ 3];
      Case aSubCode Of
        $10 : Result := Result + 'patch settings ' + CrackPatchSettings;
        $18 : Result := Result + 'module insertion';
        $21 : Result := Result + 'SEQ zoom in/out';
        Else  Result := Result + Format( '[1f PID] : unknown sub command %.2x', [ aSubCode])
      End;
    End
    Else Begin // not a channel message
      aSubCode  := aPid;
      Case aSubCode Of
        $41 : Begin
              IsSlotMsg := False;
              Result := Result + 'synth settings ' + CrackSynthSettings( False);
            End;
        $50 : Result := Result + 'new patch'
        Else  Result := Result + Format( '[1f] : unknown sub command %.2x', [ aSubCode])
      End;
    End;
  End;
Else Begin // Unknown
  Result := Format( 'unknown opcode : %.2x (in sysex : %.2x)', [ anOpcode, anOpcode Shl 2])
End;
End; // Case anOpCode Of
```

```
    Result :=
      SlotStr( IsSlotMsg) +
      PidStr ( hasPid   ) +
      ChkStr ( hasCheck ) +
      Result;
    aRaw := HexDump( aData);
  End;



end.
```