

# **PUMA Repository**

Pascal Units for Medical Applications

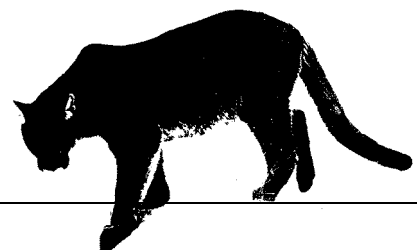
## ***HL7 engine***

---

Lazarus and Free Pascal Edition

Version 1.6  
Document version 1.6.0  
2014-08-18

Johannes W. Dietrich



The HL7 engine is a collection of Pascal units that provide functions for parsing, compiling and storing HL7 messages. Additionally, it supports reading and writing messages as files or streams.

The HL7 engine includes a basic unit (hl7.pas) and additional units that support high-level functionality like MSH or OBR segments.

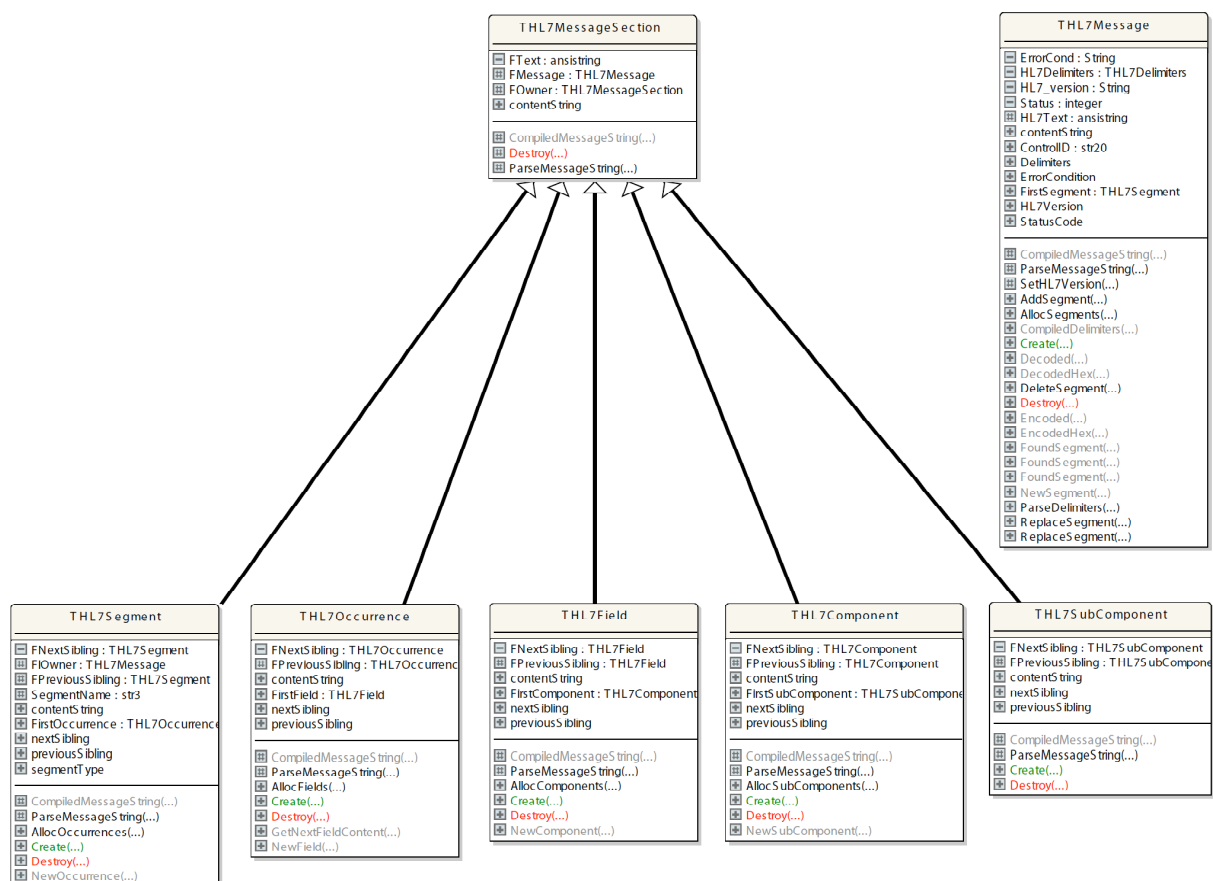


Fig. 1: Class diagram of the base unit "HL7.pas"

This documentation doesn't explain semantic meanings of HL7 fields and data types. Please refer to the official HL7 documentation for more information.

## Unit hl7.pas

The basic unit hl7.pas provides the following constants, types, data structures, classes and functions:

### Global constants

```
ksCR = #13;
ksLF = #10;
ksCRLF = #13#10;

noErr = 0;
termErr = 2;
unsuppVers = 4;
saveErr = 6;
readErr = 7;
segNotFound = 8;
createErr = 9;

STANDARD_DELIMITERS = '|^~\&';
SEGMENT_DELIMITER = ksCR;

ACKNOWLEDGEMENT_OK = 'AA';
ACKNOWLEDGEMENT_ERROR = 'AE';
ACKNOWLEDGEMENT_REJECT = 'AR';
COMMIT_ACCEPT = 'CA';
COMMIT_ERROR = 'CE';
COMMIT_REJECT = 'CR';

ESCAPE_HIGHLIGHTING = '\H\';    {start highlighting}
ESCAPE_NORMAL = '\N\';         {end highlighting}
ESCAPE_FIELD = '\F\';
ESCAPE_COMPONENT = '\S\';
ESCAPE_SUBCOMPONENT = '\T\';
ESCAPE_REPETITION = '\R\';
ESCAPE_ESCAPE = '\E\';

ESCAPE_ISO_IR6_G0 = '\C2842\';  {ISO 646 : ASCII}
ESCAPE_ISO_IR100 = '\C2D41\';   {ISO 8859 : Latin Alphabet 1}
ESCAPE_ISO_IR101 = '\C2D42\';   {ISO 8859 : Latin Alphabet 2}
ESCAPE_ISO_IR109 = '\C2D43\';   {ISO 8859 : Latin Alphabet 3}
ESCAPE_ISO_IR110 = '\C2D44\';   {ISO 8859 : Latin Alphabet 4}
ESCAPE_ISO_IR144 = '\C2D4C\';   {ISO 8859 : Cyrillic}
ESCAPE_ISO_IR127 = '\C2D47\';   {ISO 8859 : Arabic}
ESCAPE_ISO_IR126 = '\C2D46\';   {ISO 8859 : Greek}
ESCAPE_ISO_IR138 = '\C2D48\';   {ISO 8859 : Hebrew}
ESCAPE_ISO_IR148 = '\C2D4D\';   {ISO 8859 : Latin Alphabet 5}
ESCAPE_ISO_IR14 = '\C284A\';    {JIS X 0201 -1976: Romaji}
ESCAPE_ISO_IR13 = '\C2949\';    {JIS X 0201 : Katakana}
ESCAPE_ISO_IR87 = '\M2442\';    {JIS X 0208 : Kanji, hiragana and
katakana}
ESCAPE_ISO_IR159 = '\M242844\'; {JIS X 0212 : Supplementary Kanji}
```

```
ERROR_COND_MSG_ACC = '0';           {Message accepted}
ERROR_COND_SEG_SEQ_ERR = '100';      {Segment sequence error}
ERROR_COND_REQ_FLD_MISS = '101';     {Required field missing}
ERROR_COND_DATA_TYPE_ERR = '102';    {Data type error}
ERROR_COND_TBL_VAL_NOT_FND = '103';  {Table value not found}
ERROR_COND_UNSUPP_MSG_TYPE = '200';  {Unsupported message type}
ERROR_COND_UNSUPP_EVT_CODE = '201';  {Unsupported event code}
ERROR_COND_UNSUPP_PROC_ID = '202';   {Unsupported processing id}
ERROR_COND_UNSUPP_VERS_ID = '200';   {Unsupported version id}
ERROR_COND_UNKNOWN_KEY_ID = '204';   {Unknown key identifier}
ERROR_COND_DUPL_KEY_ID = '205';      {Duplicate key identifier}
ERROR_COND_APPL_REC_LOCK = '206';    {Application record locked}
ERROR_COND_INT_ERR = '207';          {Internal error}
```

```
MSH_ID = 'MSH';
```

## String types

```
str2 = string[2];
str3 = string[3];
str4 = string[4];
str5 = string[5];
str8 = string[8];
str15 = string[15];
str16 = string[16];
str20 = string[20];
str22 = string[22];
str25 = string[25];
str26 = string[26];
str40 = string[40];
str50 = string[50];
str60 = string[60];
str80 = string[80];
str180 = string[180];
str227 = string[227];
str241 = string[241];
str250 = string[250];
str427 = AnsiString;
```

## HL7 Data types

```
tCE = str250;      { HL7 CE type (Coded entry, deprecated as of HL7
v2.6) }
tCNE = AnsiString; { HL7 2.6 CNE type (Coded with no exceptions) }
tCWE = AnsiString; { HL7 2.6 CWE type (coded with exceptions) }
tCX = str250;      { HL7 CX type (Extended composite ID with check
digit) }
tCQ = AnsiString;  { HL7 CQ type (Composite quantity with units) }
tDLD = AnsiString; { HL7 DLD type (Discharge to location and date) }
tDLN = str25;      { HL7 DLN type (Driver's license number) }
tDR = str53;       { HL7 DR type (date/time range) }
tDTM = str26;      { HL7 2.7 DTM type (Date/time) }
tDT = str8;        { HL7 DT type (Date) }
tEI = str427;      { HL7 EI type (Entity identifier) }
tEIP = AnsiString; { HL7 EIP type (Entity identifier pair) }
tELD = AnsiString; { HL7 ELD type (Error location and description,
deprecated as of HL7 v2.5) }
tERL = str180;     { HL7 ERL type (Error location) }
tFC = str50;       { HL7 FC type (Financial class) }
tFT = AnsiString;  { HL7 FT type (Formatted text data) }
tHD = AnsiString;  { HL7 HD type (Hierarchic designator) }
tID = AnsiString;  { HL7 ID type (Coded value for HL7 defined
tables) }
tIS = str20;       { HL7 2.5 IS type (Coded value for user-defined
tables) }
tJCC = AnsiString; { HL7 JCC type (Job code/class) }
tMSG = str15;      { HL7 MSG type (Message type) }
tMOC = AnsiString; { HL7 MOC type (Money and charge code) }
tNDL = AnsiString; { HL7 NDL type (Name with date and location) }
tNM = str16;       { HL7 NM type (ASCII-represented number) }
tPL = str80;       { HL7 PL type (Person location) }
tPRL = AnsiString; { HL7 PRL type (Parent result link) }
tPT = str3;        { HL7 PT type (Processing type) }
tSI = str4;        { HL7 SI type (Sequence ID) }
tSPS = AnsiString; { HL7 SPS type (Specimen source, deprecated as of
HL7 v2.5) }
tST = AnsiString;  { HL7 ST type (Dstring data ) }
tTQ = str250;      { HL7 TQ type (timing/quantity, deprecated as of
HL7 v2.6) }
tTS = str26;       { HL7 2.5 TS type (Time stamp, deprecated as of
HL7 v2.6) }
tVID = str60;      { HL7 VID type (Version identifier) }
tXAD = str250;     { HL7 XAD type (Extended address) }
tXCN = str250;     { HL7 XCN type (Extended composite ID number and
name for persons) }
tXON = AnsiString; { HL7 XON type (Extended composite name and
identification number for organizations) }
tXPN = str250;     { HL7 XPN type (Extended person name) }
tXTN = str250;     { HL7 XAD type (Extended telecommunications
number) }
```

## THL7Delimiters

```
THL7Delimiters = record
  SegmentTerminator, FieldSeparator, ComponentSeparator:
    char;
  SubcomponentSeparator, RepetitionSeparator,
  EscapeCharacter: char;
end;
```

`THL7Delimiters` stores the delimiters for segments, fields, components, subcomponents and repetitions as well as the escape character.

## THL7MessageSection

```
THL7MessageSection = class
private
  {private fields}
protected
  {protected fields and methods}
public
  property contentString: ansistring;
end;
```

*THL7MessageSection* is a basic class containing mostly virtual methods and fields that are inherited by concrete implementation classes.

## THL7Segment

```
THL7Segment = class(THL7MessageSection)
protected
  {protected fields and methods}
public
  FirstOccurrence: THL7Occurrence;
  constructor Create(owner: THL7Message; SegmentText: ansistring);
  destructor Destroy; override;
  function NewOccurrence(const OccurrencesText: ansistring):
    THL7Occurrence;
  procedure AllocOccurrences(const OccurrencesText: ansistring);
  property contentString: ansistring;
  property previousSibling: THL7Segment;
  property nextSibling: THL7Segment;
  property segmentType: str3;
end;
```

*THL7Segment.FirstOccurrence* delivers the first occurrence in a segment.

*THL7Segment.NewOccurrence* creates a new occurrence.

*THL7Segment.AllocOccurrences* parses a string and creates occurrences if required (currently unimplemented).

*THL7Segment.contentString* delivers or parses the content string of the segment, depending from the direction of information flow.

*THL7Segment.previousSibling* and *THL7Segment.nextSibling* deliver the preceding or subsequent segment within the HL7 message.

*THL7Segment.segmentType* is a string defining the type of the segment (e.g. "MSH").

## THL7Occurrence

```
THL7Occurrence = class(THL7MessageSection)
protected
  {protected fields and methods}
public
  FirstField: THL7Field;
  constructor Create(owner: THL7Segment; OccurrencesText:
    ansistring);
  destructor Destroy; override;
  function NewField: THL7Field;
  function GetNextFieldContent(var aField: THL7Field): String;
  procedure AllocFields(const FieldText: ansistring);
  property contentString: ansistring;
  property previousSibling: THL7Occurrence read FPreviousSibling;
  property nextSibling: THL7Occurrence read FNextSibling;
end;
```

*THL7Occurrence.FirstField* delivers the first field in an occurrence.

*THL7Occurrence.NewField* creates a new field.

*THL7Occurrence.GetNextFieldContent* delivers the content of a field and its successor.

*THL7Occurrence.AllocFields* parses a string and creates fields if required.

*THL7Occurrence.contentString* delivers or parses the content string of the occurrence, depending from the direction of information flow.

*THL7Occurrence.previousSibling* and *THL7Occurrence.nextSibling* deliver the preceding or succeeding occurrence within the HL7 message.



## THL7Field

```
THL7Field = class(THL7MessageSection)
protected
  {protected fields and methods}
public
  FirstComponent: THL7Component;
  constructor Create(owner: THL7Occurrence; FieldText: ansistring);
  destructor Destroy; override;
  function NewComponent: THL7Component;
  procedure AllocComponents(const ComponentText: ansistring);
  property contentString: ansistring;
  property previousSibling;
  property nextSibling;
end;
```

*THL7Field.FirstComponent* delivers the first component in a field.

*THL7Field.NewComponent* creates a new component.

*THL7Field.AllocComponent* parses a string and creates components if required.

*THL7Field.contentString* delivers or parses the content string of the field, depending from the direction of information flow.

*THL7Field.previousSibling* and *THL7Field.nextSibling* deliver the preceding or succeeding field within the HL7 message.

## THL7Component

```
THL7Component = class(THL7MessageSection)
protected
  {protected fields and methods}
public
  FirstSubComponent: THL7SubComponent;
  constructor Create(owner: THL7Field; ComponentText: ansistring);
  destructor Destroy; override;
  function NewSubComponent: THL7SubComponent;
  procedure AllocSubComponents(const SubComponentText: ansistring);
  property contentString: ansistring;
  property previousSibling: THL7Component;
  property nextSibling: THL7Component;
end;
```

*THL7Component.FirstComponent* delivers the first component in a component.

*THL7Component.NewSubComponent* creates a new subcomponent.

*THL7Component.AllocSubComponent* parses a string and creates subcomponent if required.

*THL7Component.contentString* delivers or parses the content string of the component, depending from the direction of information flow.

*THL7Component.previousSibling* and *THL7Component.nextSibling* deliver the preceding or succeeding component within the HL7 message.

## THL7SubComponent

```
THL7SubComponent = class(THL7MessageSection)
protected
  {protected fields and methods}
public
  constructor Create(owner: THL7Field; ComponentText: ansistring);
  destructor Destroy; override;
  property contentString: ansistring;
  property previousSibling: THL7Component;
  property nextSibling: THL7Component;
end;
```

*THL7SubComponent.contentString* delivers or parses the content string of the subcomponent, depending from the direction of information flow.

*THL7SubComponent.previousSibling* and *THL7SubComponent.nextSibling* deliver the preceding or succeeding subcomponent within the HL7 message.

## THL7Message

```

THL7Message = class
Private
    {private fields}
protected
    {protected fields and methods}
public
    ControlID: Str20;
    FirstSegment: THL7Segment;
    procedure ParseDelimiters(DelimiterDefinition: ansistring);
    function CompiledDelimiters(const delimiters: THL7Delimiters):
        str5;
    function Encoded(const aString: ansistring): ansistring;
    function EncodedHex(const aNumber: integer): string;
    function Decoded(const aString: ansistring): ansistring;
    function DecodedHex(const aString: string): integer;
    constructor Create(version: string);
    destructor Destroy; override;
    property HL7Version: string;
    property Delimiters;
    function FoundSegment(const aSegmentName, SetID: Str3):
        THL7Segment;
    function FoundSegment(const aSegmentName, SetID: Str3;
        beginWith: THL7Segment): THL7Segment;
    function FoundSegment(const aSegmentName, SetID: Str3;
        beginWith: THL7Segment; out lastSegment: THL7Segment):
        THL7Segment;
    function NewSegment: THL7Segment;
    procedure AddSegment(theSegment: THL7Segment);
    procedure DeleteSegment(const aSegmentName, SetID: Str3);
    procedure ReplaceSegment(const aSegmentName, SetID: Str3; by:
        THL7Segment);
    procedure ReplaceSegment(const aSegmentName, SetID: Str3; by:
        THL7Segment; insertAlways: boolean);
    procedure AllocSegments(const SegmentText: ansistring);
    property contentString: ansistring;
    property StatusCode: integer;
    property ErrorCondition: string;
end;

```

*THL7Message.controlID* a unique identifier used to relate the response to the initial message.

*THL7Message.FirstSegment* delivers the first segment in a message.

*THL7Message.ParseDelimiters* reads a delimiter definition in string form and sets a THL7Delimiters record in the message.

*THL7Message.CompiledDelimiters* delivers a delimiter definition in string form from the THL7Delimiters record in the message.

*THL7Message.Encoded* encodes and escapes a string.

*THL7Message.EncodedHex* encodes a decimal number in hexadecimal form.

*THL7Message.Decoded* decodes and unescapes a string.

*THL7Message.DecodedHex* decodes a hexadecimal number and delivers a decimal representation.

*THL7Message.HL7Version* reads or sets the HL7 version (currently only version 2.5 is supported).

*THL7Message.Delimiters* reads or sets the delimiters as THL7Delimiters record.

*THL7Message.FoundSegment* delivers the first segment with name *aSegmentName* and set ID *SetID*, optionally beginning at *beginWith*.

*THL7Message.NewSegment* creates a new segment.

*THL7Message.AddSegment* creates a new segment and adds it at the end of the message's segment chain.

*THL7Message.DeleteSegment* removes the segment with name *aSegmentName* and set ID *SetID* from the segment chain.

*THL7Message.ReplaceSegment* removes the segment with name *aSegmentName* and set ID *SetID* from the segment chain and replaces it by the segment that is delivered as parameter.

*THL7Message.AllocSegments* parses a string and creates segments if required.

*THL7Message.contentString* delivers or parses the content string of the message, depending from the direction of information flow.

*THL7Message.StatusCode* delivers or defines a status code, which represents a result or an internal error message.

*THL7Message.ErrorCondition* delivers a standard HL7 error message.

## ReadHL7File

```
procedure ReadHL7File(out HL7Doc: THL7Message; const aFileName:
ansistring); overload;

procedure ReadHL7File(out HL7Doc: THL7Message; var aFile: Text);
overload;

procedure ReadHL7File(out HL7Doc: THL7Message; aStream: TStream);
overload;

procedure ReadHL7File(out HL7Doc: THL7Message; aStream: TStream;
const aBaseURI: ansistring); overload;
```

*ReadHL7File* reads an HL7 message as file or stream.

## WriteHL7File

```
procedure WriteHL7File(HL7Doc: THL7Message; const aFileName:
ansistring); overload;

procedure
WriteHL7File(HL7Doc: THL7Message; var aFile: Text); overload;

procedure WriteHL7File(HL7Doc: THL7Message; aStream: TStream);
overload;
```

*WriteHL7File* writes an HL7 message to a file or stream.

## EncodedDateTime and DecodeDateTime

```
function EncodedDateTime(DateTime: TDateTime): string;

function DecodeDateTime(StringRepresentation: string): TDateTime;
```

*EncodedDateTime* and *DecodeDateTime* encode or decode a date/time representation in HL7 format.

## Status Codes

Certain functions and procedures return status code to give indication about the result of the operation performed. Status codes are standardized for all units of the PUMA HL7 engine.

In the current version the following codes are reported:

- 0: No Error.
- 2: Block not properly terminated.
- 4: This HL7 version is not supported.
- 6: Error saving file.
- 7: Error reading file.
- 8: Segment not found.
- 9: Error creating HL7 message.

## Unit msh.pas

The unit msh.pas provides services for HL7 message headers.

### Type

```
tMSH = record
  delimiters: str5;
  sendingApp, sendingFac, receivingApp, receivingFac: tHD;
  dateTime: tDTM;
  security: str40;
  messageType: tMSG;
  controlID: str20;
  processingID: tPT;
  versionID: tVID;
  sequenceNumber: tNM;
  continuationPointer: str180;
  AccAckType, AppAckType: tID;
  countryCode: tID;
  charSet: tID;
  messageLanguage: tCE;
  altCharHandlScheme: tID;
  profileID: tEI;
  SendingRespOrg, ReceivingRespOrg: tXON; // Introduced in HL7 2.7
  SendingNetworkAddr, ReceivingNetworkAddr: tHD; // dto.
end;
```

### MSH\_Segment

```
function MSH_Segment(message: THL7Message): THL7Segment;
```

*MSH\_Segment* finds and delivers the message header segment in an HL7 message.



## GetMSH

```
procedure GetMSH(message: THL7Message; out MSHRecord: tMSH);

procedure GetMSH(message: THL7Message; out delimiters: str5; out
sendingApp, sendingFac, receivingApp, receivingFac: tHD; out
dateTime: tDTM; out security: str40; out messageType: tMSG; out
controlID: str20; out processingID: tPT; out versionID: tVID;
sequenceNumber: tNM; out continuationPointer: str180; out
AccAckType, AppAckType: tID; out countryCode: tID; out charSet:
tID; out messageLanguage: tCE; out altCharHandlScheme: tID; out
profileID: tEI);

procedure GetMSH(message: THL7Message; out delimiters: str5; out
sendingApp, sendingFac, receivingApp, receivingFac: str227; out
dateTime: str26; out security: str40; out messageType: str15; out
controlID: str20; out processingID: str3; out versionID: str60;
sequenceNumber: str15; out continuationPointer: str180; out
AccAckType, AppAckType: Str2; out countryCode: str3; out charSet:
str16; out messageLanguage: str250; out altCharHandlScheme: str20;
out profileID: str427); *
```

**GetMSH** delivers the field contents of an MSH segment.

## SetMSH

```
procedure SetMSH(message: THL7Message; aSegment: THL7Segment);

procedure SetMSH(message: THL7Message; MSHRecord: tMSH; autoDate:
boolean);

procedure SetMSH(message: THL7Message; delimiters: str5;
sendingApp, sendingFac, receivingApp, receivingFac: tHD; security:
str40; messageType: tMSG; processingID: tPT; sequenceNumber: tNM;
continuationPointer: str180; AccAckType, AppAckType: tID;
countryCode: tID; charSet: tID; messageLanguage: tCE;
altCharHandlScheme: tID; profileID: tEI);

procedure SetMSH(message: THL7Message; delimiters: str5;
sendingApp, sendingFac, receivingApp, receivingFac: tHD; dateTime:
tDTM; security: str40; messageType: tMSG; controlID: str20;
processingID: tPT; versionID: tVID; sequenceNumber: tNM;
continuationPointer: str180; AccAckType, AppAckType: tID;
countryCode: tID; charSet: tID; messageLanguage: tCE;
altCharHandlScheme: tID; profileID: tEI);

procedure SetMSH(message: THL7Message; delimiters: str5;
sendingApp, sendingFac, receivingApp, receivingFac: str227;
security: str40; messageType: str15; processingID: str3;
sequenceNumber: str15; continuationPointer: str180; AccAckType,
AppAckType: Str2; countryCode: str3; charSet: str16;
```

---

\* Deprecated method, retained for backward-compatibility only.

```
messageLanguage: str250; altCharHandlScheme: str20; profileID:  
str427); *
```

```
procedure SetMSH(message: THL7Message; delimiters: str5;  
sendingApp, sendingFac, receivingApp, receivingFac: str227;  
dateTime: str26; security: str40; messageType: str15; controlID:  
str20; processingID: str3; versionID: str60; sequenceNumber: str15;  
continuationPointer: str180; AccAckType, AppAckType: Str2;  
countryCode: str3; charSet: str16; messageLanguage: str250;  
altCharHandlScheme: str20; profileID: str427); *
```

*SetMSH* inserts an MSH segment into a message, alternatively as a predefined segment or compiled from field contents. The “autodate” variants automatically insert the current date.

## ClearMSH

```
procedure ClearMSH(var MSHRecord: tMSH);
```

*ClearMSH* clears an MSH record and sets all components to standard values.

---

\* Deprecated method, retained for backward-compatibility only.

## Unit msa.pas

The unit msa.pas provides services for HL7 message acknowledgement segments.

### Global constant

```
MSA_ID = 'MSA';
```

### Type

```
tMSA = record
  AckCode: tID;
  controlID: str20;
  textMessage: str80;
  exSeqNum: tNM;
  delAckType: char;
  ErrorCond: tCE;
  MessageWaitingNumber: tNM; // Introduced in HL7 2.7
  MessageWaitingPriority: tID; // Introduced in HL7 2.7
end;
```

### MSA\_Segment

```
function MSA_Segment(message: THL7Message): THL7Segment;
```

*MSA\_Segment* finds and delivers an acknowledgement segment in an HL7 message.

### GetMSA

```
procedure GetMSA(message: THL7Message; out MSARecord: tMSA);

procedure GetMSA(message: THL7Message; out AckCode: tID; out
controlID: str20; out textMessage: str80; out exSeqNum: tNM; out
delAckType: char; out ErrorCond: tCE);

procedure GetMSA(message: THL7Message; out AckCode: str2; out
controlID: str20; out textMessage: str80; out exSeqNum: Str15; out
delAckType: char; out ErrorCond: Str250); *
```

*GetMSA* delivers the field contents of an MSA segment.

---

\* Deprecated method, retained for backward-compatibility only.

## SetMSA

```
procedure SetMSA(message: THL7Message; aSegment: THL7Segment);  
  
procedure SetMSA(message: THL7Message; MSARecord: tMSA);  
  
procedure SetMSA(message: THL7Message; AckCode: tID; controlID:  
str20; textMessage: str80; exSeqNum: tNM; delAckType: char;  
ErrorCond: tCE);  
  
procedure SetMSA(message: THL7Message; AckCode: str2; controlID:  
str20; textMessage: str80; exSeqNum: Str15; delAckType: char;  
ErrorCond: Str250); *
```

*SetMSA* inserts an MSA segment into a message, alternatively as a predefined segment or compiled from field contents.

## ClearMSA

```
procedure ClearMSA(var MSARecord: tMSA);
```

*ClearMSA* clears a tMSA record and sets all components to standard values.

---

\* Deprecated method, retained for backward-compatibility only.

## Unit err.pas

The unit err.pas provides services for HL7 error segments.

### Global constants

```
ERR_ID = 'ERR';

SEV_WARNING = 'W';
SEV_INFO = 'I';
SEV_ERROR = 'E';
SEV_FATAL = 'F';
```

### Type

```
tERR = record
  ErrCodeLoc: tELD;
  ErrLoc: tERL;
  ErrCode: tCWE;
  severity: tID;
  appErrCode: tCWE;
  appErrPar: str80;
  DiagInfo, UserMessage: ansistring;
  InformPersIndic: tIS;
  OverrideType, OverrideReason: tCWE;
  HelpDeskContact: tXTN;
end;
```

### ERR\_Segment

```
function ERR_Segment(message: THL7Message): THL7Segment;
```

*ERR\_Segment* finds and delivers an error segment in an HL7 message.

## GetERR

```
procedure GetERR(message: THL7Message; out ERRRecord: tERR);

procedure GetERR(message: THL7Message; out ErrCodeLoc: tELD; out
ErrLoc: tERL; out ErrCode: tCWE; out severity: tID; out appErrCode:
tCWE; out appErrPar: str80; DiagInfo, UserMessage: ansistring;
InformPersIndic: tIS; OverrideType, OverrideReason: tCWE;
HelpDeskContact: tXTN);

procedure GetERR(message: THL7Message; out ErrCodeLoc, ErrLoc,
ErrCode: string; out severity: char; out appErrCode, appErrPar,
DiagInfo, UserMessage, InformPersIndic, OverrideType,
OverrideReason, HelpDeskContact: string); *
```

*GetERR* delivers the field contents of an ERR segment.

## SetERR

```
procedure SetERR(message: THL7Message; aSegment: THL7Segment);

procedure SetERR(message: THL7Message; ERRRecord: tERR);

procedure SetERR(message: THL7Message; ErrCodeLoc: tELD; ErrLoc:
tERL; ErrCode: tCWE; severity: tID; appErrCode: tCWE; appErrPar:
str80; DiagInfo, UserMessage: ansistring; InformPersIndic: tIS;
OverrideType, OverrideReason: tCWE; HelpDeskContact: tXTN);

procedure SetERR(message: THL7Message; ErrCodeLoc, ErrLoc, ErrCode:
string; severity: char; appErrCode, appErrPar, DiagInfo,
UserMessage, InformPersIndic, OverrideType, OverrideReason,
HelpDeskContact: string); *
```

The overloaded procedure *SetERR* inserts an ERR segment into a message, alternatively as a predefined segment or compiled from field contents.

## ClearERR

```
procedure ClearERR(var ERRRecord: tERR);
```

*ClearERR* clears a tERR record and sets all components to standard values.

---

\* Deprecated method, retained for backward-compatibility only.

## Unit nte.pas

The unit nte.pas provides services for HL7 notes and comments segments.

### Global constant

```
NTE_ID = 'NTE';
```

### Type

```
tNTE = record
  SetID: tSI;
  CommentSource: tID;
  comment: tFT;
  commentType: tCE;
  EnteredBy: tXCN; // Introduced in HL7 2.7
  EnteredDateTime, EffectiveStartDate, ExpirationDate: tDTM; // dto.
end;
```

### NTE\_Segment

```
function NTE_Segment(message: THL7Message): THL7Segment;
```

*NTE\_Segment* finds and delivers a notes and comments segment in an HL7 message.

### GetNTE

```
procedure GetNTE(message: THL7Message; out NTERecord: tNTE);
```

```
procedure GetNTE(message: THL7Message; out SetID: tSI; out
CommentSource: tID; out comment: tFT; out commentType: tCE);
```

```
procedure GetNTE(message: THL7Message; out SetID: str4; out
CommentSource: str8; out comment: ansistring; out commentType:
str250); *
```

*GetNTE* delivers the field contents of an NTE segment.

---

\* Deprecated method, retained for backward-compatibility only.

## SetNTE

```
procedure SetNTE(message: THL7Message; aSegment: THL7Segment);  
  
procedure SetNTE(message: THL7Message; NTERecord: tNTE);  
  
procedure SetNTE(message: THL7Message; SetID: tSI; CommentSource:  
tID; comment: tFT; commentType: tCE);  
  
procedure SetNTE(message: THL7Message; SetID: str4; CommentSource:  
str8; comment: ansistring; commentType: str250); *
```

The overloaded procedure *SetNTE* inserts an NTE segment into a message, alternatively as a predefined segment or compiled from field contents.

## ClearNTE

```
procedure ClearNTE(var NTERecord: tNTE);
```

*ClearNTE* clears a tNTE record and sets all components to standard values.

---

\* Deprecated method, retained for backward-compatibility only.



## Unit evn.pas

The unit evn.pas provides services for HL7 event type segments.

### Global constant

```
EVN_ID = 'EVN';
```

### Type

```
tEVN = record
  evtTypeCode: char;
  recDateTime, plannedDateTime: tDTM;
  reasonCode: tCWE;
  opID: tXCN;
  evtOccurred: tDTM;
  evtFacility: tHD;
end;
```

### EVN\_Segment

```
function EVN_Segment(message: THL7Message): THL7Segment;
```

*EVN\_Segment* finds and delivers an event type segment in an HL7 message.

### GetEVN

```
procedure GetEVN(message: THL7Message; out EVNRecord: tEVN);

procedure GetEVN(message: THL7Message; out evtTypeCode: char; out
  recDateTime, plannedDateTime: tDTM; out reasonCode: tCWE; out
  opID: tXCN; out evtOccurred: tDTM; out evtFacility: tHD);
```

*GetEVN* delivers the field contents of an EVN segment.

## SetEVN

```
procedure SetEVN(message: THL7Message; aSegment: THL7Segment);  
  
procedure SetEVN(message: THL7message; EVNRecord: tEVN);  
  
procedure SetEVN(message: THL7Message; evtTypeCode: char;  
recDateTime, plannedDateTime: tDTM; reasonCode: tCWE; opID: tXCN;  
evtOccurred: tDTM; evtFacility: tHD);
```

The overloaded procedure *SetEVN* inserts an EVN segment into a message, alternatively as a predefined segment or compiled from field contents.

## ClearEVN

```
procedure ClearEVN(var EVNRecord: tEVN);
```

*ClearEVN* clears a tEVN record and sets all components to standard values.

## Unit pid.pas

The unit pid.pas provides services for HL7 patient identification segments.

### Global constant

```
PID_ID = 'PID';
```

### Type

```
tPID = record
  SetID: tSI;
  PatientID: str20;
  PatientIDList: str250;
  AltPatID: str20;
  PatientName: tXPN;
  MothersMaidenName: tXPN;
  BirthDateTime: tDTM;
  AdminSex: tIS;
  PatientAlias: tXPN;
  Race: tCE;
  PatientAddress: tXAD;
  CountyCode: tIS;
  HomePhoe: tXTN; // Should be HomePhone; retained for backwards
    compatibility
  BusinessPhone: tXTN;
  PrimaryLanguage, MaritalStatus: tCE;
  Religion, PatientAccountNumber: tCE;
  SSNNumber: tST;
  DriverLicenseNumber: tDLN;
  MothersID: tCX;
  EthnicGroup: tCE;
  BirthPlace: str250;
  MultipleBirthID: tID;
  BirthOrder: tNM;
  Citizenship, VeteransMilitaryStatus: tCE;
  Nationality: tCE;
  PatientDeathDateTime: tDTM;
  PatientDeathIndicator, IDUnknownIndicator: tID;
  IDReliabilityIndicator: tIS;
  LastUpdateDateTime: tDTM;
  LastUpdateFacility: tHD;
  SpeciesCode, BreedCode: tCE;
  Strain: tST;
  ProductionClassCode: tCE;
  TribalCitizenship: tCWE;
  PatientTelecomInformation: tXTN; // Introduced in HL7 2.7
end;
```

## PID\_Segment

```
function PID_Segment(message: THL7Message): THL7Segment;
```

*PID\_Segment* finds and delivers a patient identification segment in an HL7 message.

## GetPID

```
procedure GetPID(message: THL7Message; out PIDRecord: tPID);
```

*GetPID* delivers the field contents of an PID segment.

## SetPID

```
procedure SetPID(message: THL7Message; aSegment: THL7Segment);
```

```
procedure SetPID(message: THL7message; PIDRecord: tPID);
```

The overloaded procedure *SetPID* inserts a PID segment into a message, alternatively as a predefined segment or compiled from field contents.

## ClearPID

```
procedure ClearPID(var PIDRecord: tPID);
```

*ClearPID* clears a tPID record and sets all components to standard values.

## Unit pv1.pas

The unit pv1.pas provides services for HL7 patient visit segments.

### Global constant

```
PV1_ID = 'PV1';
```

### Type

```
tPV1 = record
  SetID: tSI;
  PatientClass: tIS;
  AssignedPatientLocation: tPL;
  AdmissionType: tIS;
  PreadmitNumber: tCX;
  PriorPatientLocation: tPL;
  AttendingDoctor, ReferringDoctor, ConsultingDoctor: tXCN;
  HospitalService: tIS;
  TemporaryLocation: tPL;
  PreadmitTestIndicator, ReadmissionIndicator, AdmitSource: tIS;
  AmbulatoryStatus, VIPIndicator: tIS;
  AdmittingDoctor: tXCN;
  PatientType: tIS;
  VisitNumber: tCX;
  FinancialClass: tFC;
  ChargePriceIndicator, CourtesyCode: tIS;
  CreditRate, ContractCode: tIS;
  ContractEffectiveDate: tDT;
  ContractAmount, ContractPeriod: tNM;
  InterestCode, TransferToBadDeptCode: tIS;
  TransferToBadDeptDate: tDT;
  BadDeptAgencyCode: tIS;
  BadDeptTransferAmount, BadDeptRecoveryAmount: tNM;
  DeleteAccountIndicator: tIS;
  DeleteAccountDate: tDT;
  DischargeDisposition: tIS;
  DischargedToLocation: tDLD;
  DietType: tCE;
  ServicingFacility, BedStatus, AccountStatus: tIS;
  PendingLocation, PriorTemporaryLocation: tPL;
  AdmitDateTime, DischargeDateTime: tTS;
  CurrentPatientBalance, TotalCharges: tNM;
  TotalAdjustments, TotalPayments: tNM;
  AlternateVisitID: tCX;
  VisitIndicator: tIS;
  OtherHealthcareProvider: tXCN;
  ServiceEpisodeDescription: tST; // Introduced in HL7 2.7
  ServiceEpisodeID: tCX; // Introduced in HL7 2.7
end;
```

## PV1\_Segment

```
function PV1_Segment(message: THL7Message): THL7Segment;
```

*PV1\_Segment* finds and delivers a patient visit segment in an HL7 message.

## GetPV1

```
procedure GetPV1(message: THL7Message; out PV1Record: tPV1);
```

*GetPV1* delivers the field contents of a PV1 segment.

## SetPV1

```
procedure SetPV1(message: THL7Message; aSegment: THL7Segment);
```

```
procedure SetPV1(message: THL7message; PV1Record: tPV1);
```

The overloaded procedure *SetPV1* inserts a PV1 segment into a message, alternatively as a predefined segment or compiled from field contents.

## ClearPV1

```
procedure ClearPV1(var PV1Record: tPV1);
```

*ClearPV1* clears a tPV1 record and sets all components to standard values.

## Unit pv2.pas

The unit pv2.pas provides services for HL7 patient visit segments.

### Global constant

```
PV2_ID = 'PV2';
```

### Type

```
tPV2 = record
  PriorPendingLocation: tPL;
  AccommodationCode, AdmitReason, TransferReason: tCWE;
  PatientValuables, PatientValuablesLocation: tST;
  VisitUserCode: tCWE;
  ExpAdmitDateTime, ExpDischargeDateTime: tDTM;
  EstLOS, ActLOS: tNM;
  VisitDescription: tST;
  ReferralSourceCode: tXCN;
  PreviousServiceDate: tDT;
  EmploymentIllnessRelatedID: tID;
  PurgeStatusCode: tCWE;
  PurgeStatusDate: tDT;
  SpecialProgramCode: tCWE;
  RetentionID: tID;
  ExpNumberOfInsurancePlans: tNM;
  VisitPublicityCode: tCWE;
  VisitProtectionID: tID;
  ClinicOrgName: tXON;
  PatientStatusCode, VisitPriorityCode: tCWE;
  PreviousTreatmentDate: tDT;
  ExpDischargeDispo: tCWE;
  SignatureOnFileDate, FirstSimilarIllnessDate: tDT;
  PatientChargeAdjustmentCode, RecurringServiceCode: tCWE;
  BillingMediaCode: tID;
  ExpSurgeryDateTime: tDTM;
  MilPartnershipCode, MilNonAvailCode: tID;
  NewbornBabyID, BabyDetainedID: tID;
  ModeOfArrivalCode, RecreationalDrugUseCode: tCWE;
  AdmissionLevelOfCareCode, PrecautionCode: tCWE;
  PatientConditionCode: tCWE;
  LivingWillCode, OrganDonorCode, AdvanceDirectiveCode: tCWE;
  PatientStatusEffectiveDate: tCWE;
  ExpectedLOAReturnDateTime: tDTM;
  ExpectedPreadmissionTestingDateTime: tDTM;
  NotifyClergyCode: tID;
  AdvanceDirectiveLastVerifiedDate: tDT; // Introduced in HL7 2.6
end;
```

## PV2\_Segment

```
function PV2_Segment(message: THL7Message): THL7Segment;
```

*PV2\_Segment* finds and delivers a patient visit segment in an HL7 message.

## GetPV2

```
procedure GetPV2(message: THL7Message; out PV2Record: tPV2);
```

*GetPV2* delivers the field contents of a PV2 segment.

## SetPV2

```
procedure SetPV2(message: THL7Message; aSegment: THL7Segment);
```

```
procedure SetPV2(message: THL7message; PV2Record: tPV2);
```

The overloaded procedure *SetPV2* inserts a PV2 segment into a message, alternatively as a predefined segment or compiled from field contents.

## ClearPV2

```
procedure ClearPV2(var PV2Record: tPV2);
```

*ClearPV2* clears a tPV2 record and sets all components to standard values.



## Unit nk1.pas

The unit nk1.pas provides services for HL7 next of kin segments.

### Global constant

```
NK1_ID = 'NK1';
```

### Type

```
tNK1 = record
  SetID: tSI;
  Name: tXPN;
  Relationship: tCE;
  Address: tXAD;
  PhoneNumber, BusinessPhoneNumber: tXTN;
  ContactRole: tCE;
  StartDate, EndDate: tDT;
  JobTitle: tST;
  JobCode: tJCC;
  EmployeeNumber: tCX;
  OrganizationName: tXON;
  MaritalStatus: tCE;
  AdministrativeSex: tIS;
  DateTimeOfBirth: tTS;
  LivingDependency, AmbulatoryStatus: tIS;
  Citizenship, PrimaryLanguage: tCE;
  LivingArrangement: tIS;
  PublicityCode: tCE;
  ProtectionIndicator: tID;
  StudentIndicator: tIS;
  Religion: tCE;
  MothersMaidenName: tXPN;
  Nationality, EthnicGroup, ContactReason: tCE;
  ContactPersonsName: tXPN;
  ContactPersonsTelephoneNumber: tXTN;
  ContactPersonsAddress: tXAD;
  Identifiers: tCX;
  JobStatus: tIS;
  Race: tCE;
  Handicap: tIS;
  ContactPersonSocialSecurityNumber, BirthPlace: tST;
  VIPIndicator: tIS;
  TelecomInformation, ContactPersonsTelecomInformation: tXTN;
  // Introduced in HL7 2.7
end;
```

## NK1\_Segment

```
function NK1_Segment(message: THL7Message): THL7Segment;
```

*NK1\_Segment* finds and delivers a next of kin segment in an HL7 message.

## GetNK1

```
procedure GetNK1(message: THL7Message; out NK1Record: tNK1);
```

*GetNK1* delivers the field contents of an NK1 segment.

## SetNK1

```
procedure SetNK1(message: THL7Message; aSegment: THL7Segment);
```

```
procedure SetNK1(message: THL7message; NK1Record: tNK1);
```

The overloaded procedure *SetNK1* inserts an NK1 segment into a message, alternatively as a predefined segment or compiled from field contents.

## ClearNK1

```
procedure ClearNK1(var NK1Record: tNK1);
```

*ClearNK1* clears a tNK1 record and sets all components to standard values.

## Unit obr.pas

The unit obr.pas provides services for HL7 observation request segments.

### Global constant

```
OBR_ID = 'OBR';
```

### Type

```
tOBR = record
  SetID: tSI;
  PlacOrdNumb, FillOrdNumb: tEI;
  USI: tCE; { Universal Service Identifier }
  Priority: tID;
  ReqDateTime, ObsDateTime, ObsEndDateTime: tDTM;
  CollectionVolume: tCQ;
  CollectorIdentifier: tXCN;
  SpecimenActionCode: tID;
  DangerCode: tCE;
  RelevantClinicalInfo: tST;
  SpecimenReceivedDateTime: tTS;
  SpecimenSource: tSPS;
  OrderingProvider: tXCN;
  OrderCallbackPhoneNumber: tXTN;
  PlacerField1, PlacerField2: ansistring;
  FillerField1, FillerField2: ansistring;
  ResultsRptStatusChng: tTS;
  ChargeToPractice: tMOC;
  DiagnosticServSectID, ResultStatus: tID;
  ParentResult: tPRL;
  QuantityTiming: tTQ;
  ResultCopiesTo: tXCN;
  Parent: tEIP;
  TransportationMode: tID;
  ReasonForStudy: tCE;
  PrincipalResultInterpreter, AssistantResultInterpreter: tNDL;
  Technician, Transcriptionist: tNDL;
  ScheduledDateTime: tTS;
  NumberOfSampleContainers: tNM;
  TransportLogisticsOfCollSampl, CollectorsComment: tCE;
  TransportArrangementResponsibility: tCE;
  TransportArranged, EscortRequired: tID;
  PlannedPatientTransportComment: tCE;
  ProcedureCode, ProcedureCodeModifier: tCE;
  PlacerSupplServiceInfo, FillerSupplServiceInfo: tCE;
  MedicallyNecessaryDuplProcReason: tCWE;
  ResultHandling: tIS; // Introduced in HL7 2.7
  ParentUniversalServiceID: tCWE; // Introduced in HL7 2.7
  ObservationGroupID, ParentObservationGroupID: tEI; // dto.
```

```
AltPlacerOrderNumber: tCX; // Introduced in HL7 2.7  
end;
```

## OBR\_Segment

```
function OBR_Segment(message: THL7Message): THL7Segment;
```

*OBR\_Segment* finds and delivers an observation request segment in an HL7 message.

## GetOBR

```
procedure GetOBR(message: THL7Message; out OBRRecord: tOBR);  
  
procedure GetOBR(message: THL7Message; out SetID: tSI; out  
PlacOrdNumb, FillOrdNumb: tEI; out USI: tCE; out Priority: tID;  
out ReqDateTime, ObsDateTime, ObsEndDateTime: tDTM); *  
  
procedure GetOBR(message: THL7Message; out SetID: str4; out  
PlacOrdNumb, FillOrdNumb: str22; out USI: str250; out Priority:  
Str2; out ReqDateTime, ObsDateTime, ObsEndDateTime: str26); *
```

*GetOBR* delivers the field contents of an OBR segment.

## SetOBR

```
procedure SetOBR(message: THL7Message; aSegment: THL7Segment);  
  
procedure SetOBR(message: THL7Message; OBRRecord: tOBR);  
  
procedure SetOBR(message: THL7Message; SetID: tSI; PlacOrdNumb,  
FillOrdNumb: tEI; USI: tCE; Priority: tID; ReqDateTime, ObsDateTime,  
ObsEndDateTime: tDTM); *  
  
procedure SetOBR(message: THL7Message; SetID: str4; PlacOrdNumb,  
FillOrdNumb: str22; USI: str250; Priority: Str2; ReqDateTime,  
ObsDateTime, ObsEndDateTime: str26); *
```

The overloaded procedure *SetOBR* inserts an OBR segment into a message, alternatively as a predefined segment or compiled from field contents.

---

\* Deprecated method, retained for backward-compatibility only.

**ClearOBR**

```
procedure ClearOBR(var OBRRecord: tOBR);
```

*ClearOBR* clears a tOBR record and sets all components to standard values.

## Unit obx.pas

The unit obx.pas provides services for HL7 observation / result segments.

### Global constant

```
OBX_ID = 'OBX';
```

### Type

```
tOBX = record
  SetID: tSI;
  ValueType: tID;
  ObsID: tCE;
  obsSubID: tST;
  obsValue: ansistring;
  Units: tCE;
  RefRange: tST;
  AbnormFlags: tIS; // referred to as Interpretation Codes
    in HL7 2.7
  probability: tNM;
  Nature, status: tID;
  RRDate: tDTM;
  UDAC: tST;
  ObsDateTime: tDTM;
  prodID: tCE;
  respObs: tXCEN;
  observMethod: tCE;
  EquipInstID: tEI;
  AnalysisDateTime: tDTM;
  ObservationSite: tCWE; // Introduced in HL7 2.7
  ObservationInstanceID: tEI; // Introduced in HL7 2.7
  MoodCode: tCNE; // Introduced in HL7 2.7
  PerformingOrgName: tXON; // Introduced in HL7 2.7
  PerformingOrgAddr: tXAD; // Introduced in HL7 2.7
  PerformingOrgMedicalDirector: tXCEN; // Introduced in HL7 2.7
  PatientResultsReleaseCat: tID; // Introduced in HL7 2.7
end;
```

### OBX\_Segment

```
function OBX_Segment(message: THL7Message): THL7Segment;
```

*OBX\_Segment* finds and delivers an observation / result segment in an HL7 message.

## GetOBX

```
procedure GetOBX(message: THL7Message; out OBXRecord: tOBX);

procedure GetOBX(message: THL7Message; out SetID: tSI; out
ValueType: tID; out ObsID: tCE; obsSubID: tST; out obsValue:
ansistring; out Units: tCE; out RefRange: tST; AbnormFlags: tIS; out
probability: tNM; out Nature, status: tID; out RRDate: tDTM; UDAC:
tST; out ObsDateTime: tDTM; out prodID: tCE; respObs: tXCN;
observMethod: tCE; EquipInstID: tEI; out AnalysisDateTime: tDTM);

procedure GetOBX(message: THL7Message; out SetID: str4; out
ValueType: str2; out ObsID: str250; obsSubID: str20; out obsValue:
AnsiString; out Units: str250; out RefRange: str60; AbnormFlags,
probability: str5; out Nature: str2; out status: char; out RRDate:
str26; UDAC: str20; out ObsDateTime: str26; out prodID, respObs,
observMethod: str250; EquipInstID: str22; out AnalysisDateTime:
str26); *
```

**GetOBX** delivers the field contents of an OBX segment.

## SetOBX

```
procedure SetOBX(message: THL7Message; aSegment: THL7Segment);

procedure SetOBX(message: THL7Message; OBXRecord: tOBX);

procedure SetOBX(message: THL7Message; SetID: tSI; ValueType: tID;
ObsID: tCE; obsSubID: tST; obsValue: ansistring; Units: tCE;
RefRange: tST; AbnormFlags: tIS; probability: tNM; Nature, status:
tID; RRDate: tDTM; UDAC: tST; ObsDateTime: tDTM; prodID: tCE;
respObs: tXCN; observMethod: tCE; EquipInstID: tEI;
AnalysisDateTime: tDTM);

procedure SetOBX(message: THL7Message; SetID: str4; ValueType: str2;
ObsID: str250; obsSubID: str20; obsValue: AnsiString; Units: str250;
RefRange: str60; AbnormFlags, probability: str5; Nature: str2;
status: char; RRDate: str26; UDAC: str20; ObsDateTime: str26;
prodID, respObs, observMethod: str250; EquipInstID: str22;
AnalysisDateTime: str26); *
```

**SetOBX** inserts an OBX segment into a message, alternatively as a predefined segment or compiled from field contents.

---

\* Deprecated method, retained for backward-compatibility only.

**ClearOBX**

```
procedure ClearOBX(var OBXRecord: tOBX);
```

*ClearOBX* clears a tOBX record and sets all components to standard values.



## Unit spm.pas

The unit spm.pas provides services for HL7 specimen segments.

### Global constant

```
SPM_ID = 'SPM';
```

### Type

```
tSPM = record
  SetID:          tSI;
  SpecimenID, SpecimenParentID: tEIP;
  SpecimenType, SpecimenTypeMod: tCWE;
  SpecimenAdditives, SpecimenCollectionMethods: tCWE;
  SpecimenSourceSite, SpecimenSourceSiteMod: tCWE;
  SpecimenCollectionSite, SpecimenRole: tCWE;
  SpecimenCollectionAmount: tCQ;
  GroupedSpecimenCount: tNM;
  SpecimenDescription: tST;
  SpecimenHandlingCode, SpecimenRiskCode: tCWE;
  SpecimenCollectionDateTime: tDR;
  SpecimenReceivedDateTime, SpecimenExpirDateTime: tDTM;
  SpecimenAvailability: tID;
  SpecimenRejectReason, SpecimenQuality: tCWE;
  SpecimenAppropriateness, SpecimenCondition: tCWE;
  SpecimenCurrentQuality: tCQ;
  NumberOfSpecimenContainers: tNM;
  ContainerType, ContainerCondition: tCWE;
  SpecimenChildRole: tCWE;
  AccessionID, otherSpecimenID: tCX;
  ShipmentID: tEI
end;
```

### SPM\_Segment

```
function SPM_Segment(message: THL7Message): THL7Segment;
```

*SPM\_Segment* finds and delivers a patient visit segment in an HL7 message.

### GetSPM

```
procedure GetSPM(message: THL7Message; out SPMRecord: tSPM);
```

*GetSPM* delivers the field contents of an SPM segment.

## SetSPM

```
procedure SetSPM(message: THL7Message; aSegment: THL7Segment);
```

```
procedure SetSPM(message: THL7message; SPMRecord: tSPM);
```

The overloaded procedure *SetSPM* inserts an SPM segment into a message, alternatively as a predefined segment or compiled from field contents.

## ClearSPM

```
procedure ClearSPM(var SPMRecord: tSPM);
```

*ClearSPM* clears a tSPM record and sets all components to standard values.

## Unit mllp.pas

The unit mllp.pas is a support unit providing MLLP (minimal lower layer protocol).

### Global constants

```
SB = chr($B);  
EB = chr($1c);
```

### Type

```
TMLLP = class(TObject)  
private  
    {private fields}  
protected  
    {protected fields and methods}  
public  
    constructor Create;  
    destructor Destroy;  
    property message: THL7Message;  
    property block: ansistring;  
    property StatusCode: integer;  
end;
```

*TMLLP.message* the message that is encoded via MLLP.

*TMLLP.block* message encoded as MLLP block.

*TMLLP.StatusCode* delivers or defines a status code, which represents a result or an internal error message.

© J. W. Dietrich, 1994 – 2014  
© Ludwig Maximilian University of Munich 1995 – 2002  
© University of Ulm Hospitals 2002 – 2004  
© Ruhr University of Bochum 2005 – 2014

Source code released under the BSD License

HL7 and Health Level Seven are registered trademarks of Health Level Seven International. Reg. U.S. Pat & TM Off.

Title image modified from Wikimedia Commons  
(<http://commons.wikimedia.org/wiki/File:Mountain-lion-01623.jpg>)

<http://puma-repository.sf.net>