

PUMA Repository

Pascal Units for Medical Applications

HL7 engine

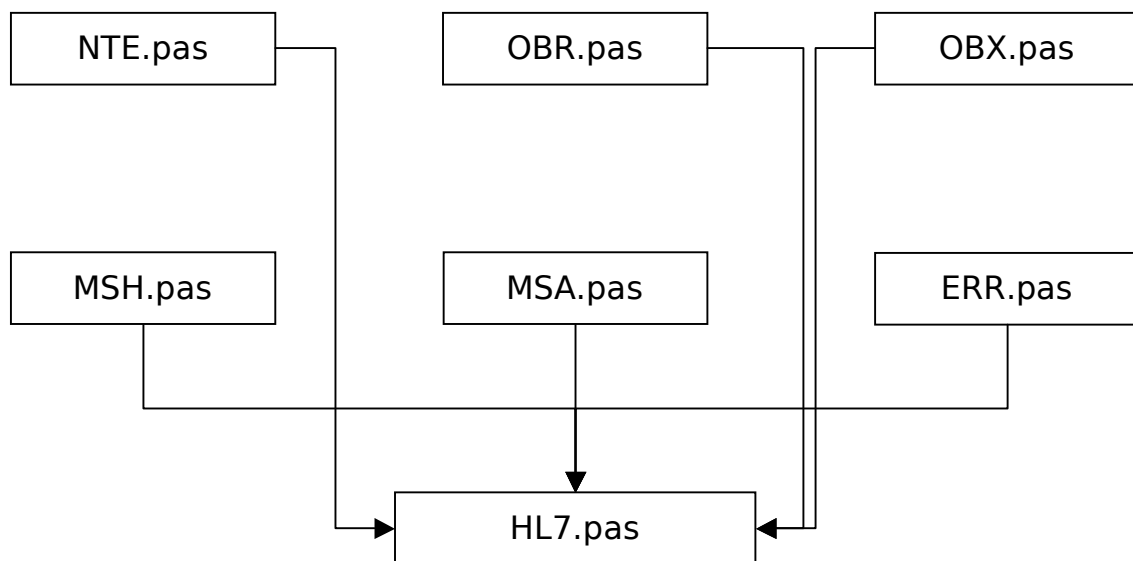
Lazarus and Free Pascal Edition

Version 1.2
Document version 1.2.0
2013-12-23

Johannes W. Dietrich

The HL7 engine is a collection of Pascal units that provide functions for parsing and compiling 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.



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;

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}

MSH_ID = 'MSH';
```

String types

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

THL7Delimiters

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

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;
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.

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.

Unit msh.pas

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

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 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 a MSH segment.

SetMSH

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

```
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;
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 a MSH segment into a message, alternatively as a predefined segment or compiled from field contents.

Unit msa.pas

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

Global constant

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

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 AckCode: str2; out  
controlID: str20; out textMessage: str80; out exSeqNum: Str15; out  
delAckType: char; out ErrorCond: Str250);
```

GetMSA delivers the field contents of a MSA segment.

SetMSA

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

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

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

Unit err.pas

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

Global constant

```
ERR_ID = 'ERR';
```

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 ErrCodeLoc, ErrLoc,
ErrCode: string; out severity: char; out appErrCode, appErrPar,
DiagInfo, UserMessage, InformPersIndic, OverrideType,
OverrideReason, HelpDeskContact: string);
```

GetMSA delivers the field contents of an ERR segment.

SetERR

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

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.

Unit nte.pas

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

Global constant

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

NTE_Segment

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

NTE_Segment finds and delivers an notes and comments segment in an HL7 message.

GetNTE

```
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.

SetNTE

```
procedure SetNTE(message: THL7Message; aSegment: THL7Segment);  
  
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.

Unit obr.pas

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

Global constant

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

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 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; SetID: str4; PlacOrdNumb,  
FillOrdNumb: str22; USI: str250; Priority: Str2; ReqDateTime,  
ObsDateTime, ObsEndDateTime: str26);
```

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

Unit obx.pas

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

Global constant

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

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 SetID: str4; out  
Value: str2; out ObsID: str250; out 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; SetID: str4; Value: 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.

© J. W. Dietrich, 1994 – 2013
© Ludwig Maximilian University of Munich 1995 – 2002
© University of Ulm Hospitals 2002 – 2004
© Ruhr University of Bochum 2005 – 2013

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.

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