Chapter 1
Introduction

This part presents the syntax used for expressing in an unambiguous way the components of the messages, and the special functions used for achieving the validation of an individual component. This checking is independent of the sequence of components.

For this purpose, these chapters have been identified:

- Chapter 2 gives the precise definition of a component inside a S.W.I.F.T. message.
- Chapter 3 describes the conventions used for the notation of the components.
- Chapters 4 and 5 list the syntax and the purpose of the special functions used for the validation of the components.
- Chapter 6 documents the Message User Group Text Validation Rules.

Chapter 2
Definition

A component is a character or string of characters which is defined as a logical subdivision of a text field. A text field may consist of single or multiple components as specified here. Multiple components may be semantically gathered. In case such a group of components does not define precisely the entire field, it will be called 'subfield'.

There are three general categories of components:

- a component of variable length which consists of all numerics, all alphabetics, all alphanumerics, or any characters of the permissible S.W.I.F.T. character sets except 'CRLF', '/', '//' characters which are primarily used as component separators

  Each particular occurrence of 'CRLF', '/', '//' will be signaled.
- a component of fixed length where the order of each character is predefined or belongs to a given list
- a component of variable length which is defined by a specific validation rule mixing numeric and special characters, e.g., the amount component which consists of numerics and a unique separator.

Exceptions:

- meaningless components are not allowed: components containing only blank(s) or only 'CrLf' are not allowed, with the exception of some special fields. (refer to Part II Chapter 5 for field 77E, 77T exceptions and examples)

  Examples of meaningless (invalid) components in field 72 of an MT200:

:72:'CrLf'	1st subfield is empty
//ABCDXYZ'CrLf'	
:72:/ACC/ee'CrLf'	2nd component contains only blanks
//ABCDXYZ'CrLf'	
:72:/ACC/'CrLf'	2nd subfield, component contains only blanks
//ee'CrLf'	
:72:/ACC/ABCDEFG'CrLf'	2nd subfield, component is empty
//'CrLf'	

A component will be validated using one of the three major functions as follows:

- The component form must be one of the elements of a predefined set (code words). e.g., the component is a 'D' or 'C' denoting a debit or credit transaction. All code words must be in upper case.
- The component form is constructed from a composition of elements from predefined set(s).
- The component form is constructed from a composition of elements from predefined set(s) and the meaning of this component corresponds to some semantic checking requirements.

The component validation requirements will be specified on a functional basis, i.e., all of the functions needed to validate all currently existing components will be defined. In order to precisely specify the validation functions, the method utilised to present the functions

will incorporate both a narrative description of each validation function and a representation faithful to the formatting rules defined in *General Information - Standards*.

The component validation functions specified herein will subsequently be used to define the field validation requirements based on component composition.

Chapter 3
Conventions

As specified in Chapter 2, the components validation requirements will be defined on a functional basis, using standard representation conventions given in the *S.W.I.F.T. User Handbook*.

The main notation conventions used in this document are:

- Variable components values are printed between angle brackets (<>).

  For example: <AMOUNT>, <NUMBER>, <DATE1>, etc.

- Where a component or a subfield is optional, it is printed between square brackets ([ ]).

  For example: [<DATE1>], ['/' <DC>], etc.

- If a component, a group of components, or a subfield, may be repeated several times, the minimum and maximum number of repetitions is specified immediately after the definition of that entity.

  For example:

  ['CRLF'35x]0-3

  the group <carriage return _ line feed _ up-to-35-characters> is optional, and may be repeated maximum 3 times.
  Note: in this case we could have used parentheses instead of square brackets, the result would be identical.

  <DATE2>['/'<DATE1>]0-11

  the component <DATE2> is mandatory, the subfield ['/'<DATE1>] is optional, but it may be present up to 11 times.

- A mandatory choice of one component or subfield from several possibilities is indicated by a vertical bar (|, the vertical bar represents the exclusive OR relationship).

  For example:

  <HHMM> | <TIME2>

  either <HHMM> or <TIME2> must be present, not both.

  7!a<DATE2> | <HHMM>

  this subfield must be composed of a 7 alphabetic character component (fixed length), followed by either <DATE2> or <HHMM> (one and only one of the later 2 components must be present).

- Parentheses may be used to identify groups of components that belong together, they may also be required to specify an OR (|) relationship between groups of components.

  For example: ref. <FLD72> in Part II, section 4.10. ['CRLF'(<INSTR>|'//'33x)]0-5

  Where each of up to 5 optional lines is/are validated by the function <INSTR> (ref. Part II, 4.9), or the line must begin with '//' (i.e. the first 2 characters are a double slash) and must contain minimum 1 character, maximum 33 characters from the <x> character set.

The full rules for the notation of components inside messages and fields can be found in the *S.W.I.F.T. User Handbook*.

These rules can be summarized as follows:

Field Length		Field Type	
nn	: maximum length (minimum is 1)	n	: numeric digits (0 through 9) only
nn-nn	: minimum and maximum length	a	: alphabetic capital letters (A through Z), upper case only
nn!	: fixed length	x	: SWIFT X set any character of the X permitted set (General FIN application set) upper case and lower case allowed
nn*nn	: maximum number of lines times maximum line length	y	: SWIFT Y set any character of the Y permitted set (EDI service specific set), upper case only
		z	: SWIFT Z set all characters included in the X and Y sets, plus a couple of special characters
		c	: alpha-numeric capital letters (upper case), and digits only
		h	: hexadecimal letters A through F (upper case), and digits only
		e	: blank or space
		A	: alphabetic, upper case or lower case A through Z, a through z
		B	: alphanumeric upper case or lower case A through Z, a through z and 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Based on these rules, the following sets will be used for the presentation of the validation functions:

<DC>	'D' | 'C'	the debit/credit code (Error Code T51)
<DM>	'D' | 'M'	the days/months code (Error Code T61)
<SIGN>	'+' | '-'	the plus/minus code (Error Code T15)

The value represented by the following set is maintained in a subtable in the currency code table (CURSEP). The maximum number of decimal digits permitted for each currency code is also included in this table.

<CUR> 3!a = must be a valid ISO 4217 currency code, e.g., USD for the United States dollar.(Error Code T52)

This table lists the currency codes that are used to validate field components declared as currencies (their format <CUR> is to be validated against the codes included in this table).

All code words (e.g., 'D', 'C', 'PCT', 'OUR', 'BEN', etc) including currency codes ('USD', 'EUR', 'BEF', etc.) must be in upper case format.

When <CUR> is validated as: "a 3 char code word, or a valid currency code", it is documented in Part III, chapter 4, e.g. field 37K in MT 305, 306, and common group, quote:

37	K	1		305 306 *	'PCT' or a valid currency code	T52

The term "BIC" is used to refer to the ISO Bank Identifier Code; depending on where it is used in a FIN message, a BIC must be either connected (referred to as a SWIFT BIC) or not connected (referred to as a non-SWIFT BIC).

FIN user-to-user messages may only be sent from and received by SWIFT connected users, therefore, only SWIFT BICs can be used in the header of a SWIFT FIN message, while SWIFT BICs and non-SWIFT BICs may be referenced in the text of the FIN messages (refer to Part III, Chapter 3, fields formats).

Additional content checks are performed. Refer to the special field exceptions table for a listing of values used for code word checking (text validation).

Chapter 4
Special functions

4.1 Date

The **Date** function checks the date component, the function is represented by:

<DATE1>, or <DATE2>, or <DATE4>

The syntax for each function is as follows:

<DATE1> MMDD

<DATE2> YYMMDD

<DATE4> YYYYMMDD

Where all ’Y’, ’M’, and ’D’ characters must be numeric characters :

i.e. figures 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

'YYYY' must not consist entirely of zeroes, e.g. '0000' is rejected "T50".

’MM’ check: eg., 00<MM<13 where ’MM’ cannot exceed 12 or be less than 01.

’DD’ check: this check depends upon the ’YY’ or ’YYYY’, and ’MM’ values.

eg., 00<DD<32: check for a 31 day month.

eg., 00<DD<31: check for a 30 day month

Leap years are accommodated in the validation check:

eg., 00<DD<30: for a leap year 29-day month

eg., 00<DD<29: for a year having a 28-day month.

Standard presentation: <DATE1> 4!n
<DATE2> 6!n
<DATE4> 8!n

<DATE1> validation specs.

Depending on the field where the <DATE1> is recorded, the value of the ”year” is selected as follows:

- in field 61: the year value is selected from the system date at the time the validation is performed.

The code error ”T50” is returned after a format error in the (date) components <DATE1>, or <DATE2>, or <DATE4>.

4.1.1 Value Date

The Value Date function allows the receiver to request some FIN messages in Value Date order. At present, only the following message types are considered for Value Date ordering:

MT910 (field 32A),

all category 1 and category 2 messages containing fields 30 or 32A, or both 30 and 32A, except the common group messages 192, 292, 195, 295, 196 and 296.

List of Message types candidates for Value Date Ordering:

Cat 1	Cat 2	Cat 9
	200	910
101	201	
102_not_STP	202, 202COV	
102_STP	203	
103_not_STP	204	
103_STP	205, 205COV	
104		
107	207	
110	210	
111	256	
112		

The valid range of value dates implemented on the SWII system is from 1980 to 2060 (limits included), therefore the <year> (”YY” in <DATE2> format) of a value date component must be included in the allowed value date range.

The ”YY” component of a Value Date <DATE2> component is validated as follows:

```
IF YY >79
THEN year = ”19” || ”YY” (the sign ”||” means ”concatenate”)
ELSE year = ”20” || ”YY”
ENDIF
IF year > 2060
THEN <error:’T50’>
ENDIF
```

Note: *This validation is in addition to the <DATE2> format validation.*

If there is more than one Value Date Field in a message, for Value Date ordering purpose, the lesser date will be selected. (ref. example below)

EX: MTxxx

...

:30:951218 Value Date = 18 December 1995

...

:32A:020103USD123000,00 Value Date = 3 January 2002

...

In this example the Field 30 Value Date (18 December 1995) is selected for Value Date ordering of the message.

The code error ”T50” is returned after an invalid value date.

4.1.2 YEAR

The function **YEAR** is represented in the fields definitions by : **<YEAR>**.

The syntax is : 4!n

<4!n> must not consist entirely of zeroes, e.g. '0000' is rejected "T50".

4.2 Time

The **Time** function checks the time component, the function is represented by:

<HHMM>, <TIME2>, or <TIME3>.

The syntax for the <TIME2> function is: HHMMSS.

The syntax for the <TIME3> function is: HH[MM].

The time sub-components are validated as follows:

00<=HH<=23

00<=MM<=59

00<=SS<=59

Standard presentation:

<HHMM> 4!n
<TIME2> 6!n
<TIME3> 2!n[2!n]

The error code"T38" is returned after a format error in the (time) components <HHMM>, or <TIME2>.

The error code"T39" is returned after a format error in the (time) component <TIME3>; refer fields 98D and 98E UTC indicator.

4.3 <SWIFTBIC> and <NON-SWIFTBIC> used in copy field(s) of MTs n92, n95, n96

In the copy fields of the Message Types n92, n95 and n96 the functions <SWIFTBIC> and <NON-SWIFTBIC> perform the following syntax checks:

Bank code : 4!a	four alphabetic characters (fixed length).
Country code : 2!a	two alphabetic characters (fixed length).
Location code 1 : 1!c	one alphanumeric character, digits '0' and '1' not allowed.
Location code 2 : 1!c	one alphanumeric character, letter 'O' not allowed.
Branch code : [3!c]	optional, three alphanumeric characters.

The code 'BIC' is not allowed.

The letter 'X' is not allowed as the first letter, unless it is used in 'XXX'.

A Test and Training destination is not allowed if the emitter of the message is a LIVE user.

If an error is detected, the system returns the error code T27.

T27 - BIC incorrectly formatted or invalid.

4.4 <SWIFTBIC> and <NON-SWIFTBIC> used in other than: copy field(s) of MTs n92, n95, n96

4.4.1 S.W.I.F.T. Address: <SWIFTBIC>

This function is performed because the 8th position of the BIC address does NOT contain the numeric character ”1” value.

The function <SWIFTBIC> checks if the component consists of 8 alphanumeric characters, or 8 alphanumeric characters followed by 3 optional alphanumeric characters. The first eight characters representing a SWIFT DESTINATION are in the format 6!a<LC> (ref. Standards General Information, section 3.4.2 : <LC> = Location Code = 2!c). Therefore, the component is variable-length and may consist of eight or eleven characters.The first eight characters must represent a S.W.I.F.T. FIN DESTINATION, i.e. a destination pre-defined in the S.W.I.F.T. Database, and enabled for the FIN application, and with the status ”TRUE-CUTOVER” (unpublished SWIFTBICs are not allowed in this text part of the FIN messages). The last three optional characters must be one of the BRANCH CODES defined in the db for that destination.

Note: *Since all SWIFTBICs are published in the BIC directory with an 8 character address, the branch code ”XXX” is assigned to all SWIFTBICs, however the branch code ”XXX” is not printed in the BIC directory.*

Note: *A published 8 character SWIFT BIC with an unpublished branch code is ACK, the system does not have a flag to recognize the unpublished branch codes; therefore, only the unpublished destinations (8 char SWIFTBIC) are rejected.*

Standard presentation: <SWIFTBIC>::=<SWIFTDESTINATION>[3!c]

<SWIFTDESTINATION>::= 8 char. dest. recorded in the SWII db. as either:

1. a Master destination with the statuses ”enabled for FIN” and ”officially cutover”, or
2. a Synonym destination with the status ”enabled for FIN”, and its Master with the status ”officially cutover”, or
3. a Test and Training destination with the status ”enabled for FIN”.

A Test and Training destination is not allowed if the emitter of the message is a LIVE user.

4.4.2 NON-S.W.I.F.T. Address: <NON-SWIFTBIC>

This function is performed because the 8th position of the BIC address contains the numeric character ”1” value.

The function <NON-SWIFTBIC> checks if the component consists of 8 alphanumeric characters, or 8 alphanumeric characters followed by 3 alphanumeric characters. The first 8 characters must represent a BIC address pre-defined in the S.W.I.F.T. database as a non-connected SWIFT institution.

If the BIC is 11 char long, the last three characters must be one of the BRANCH CODES defined in the db for that 8 char. destination.

Note: *Similar to the SWIFTBICs, the branch code ”XXX” is assigned (by default) to the NON-SWIFTBIC that is published as an 8 character address in the BIC directory. However all NON-SWIFTBICs are not necessarily published as an 8 char address, hence the branch code ”XXX” is not necessarily valid for some NON-SWIFTBICs.*

Standard presentation:

<NON-SWIFTBIC>::=<7!c> ”1” [<3!c>]

<NON-SWIFTBIC>::= 8 or 11 char. dest. recorded in the SWII db. as a not-connected (for FIN) financial institution.

Note: about NON-SWIFTBIC's branch codes

The branch code for NON-SWIFTBIC's is not always optional. A NON-SWIFTBIC is published as either: (1) an 8 char. address, or (2) one or more 11 char. addresses, or (3) an 8 char. address and one or more 11 char. addresses. It is only when the NON–SWIFTBIC is published as an 8 char. address, with or without 11 char. address(es), that the branch code "XXX" may be used, i.e. is optional (ref. 1 or 3), the published 8 char. address alone may be used (ref. 1 or 3), or any of the published 11 char. addresses may be used (ref. 3).
By the same token, if a NON-SWIFTBIC is not published as an 8 char. address, i.e. is published as one or more 11 char. addresses, then the user is not allowed to refer to the 8 char. address alone, nor to use the branch code "XXX"; but the user must necessarily refer to one of the published 11 char. addresses (ref. 2).

4.5 Message Type

The function Message Type (or <MT>) will validate a component containing a message type according to the following rule: the component must be composed of 3 numeric characters, and its value must be greater than or equal to 100, and less than or equal to 999.

The error code generated by <MT> will be T18.

4.6 <Number><Amount>

These 2 functions check a number component which is of variable length.

An invalid <Number> or <Amount> component will be rejected with the error code T40 or T43.

The <Number> and <Amount> components consists of multiple parts defined as follows:

- An integer part which consists of one to n numerics, where n is dependent on the maximum length specified for the component and dependent on the number of characters in the decimal part (if present). The integer part is mandatory in the number component and at least one character must appear, leading zeros are allowed.
- A decimal separator part consisting of one special character which must be a decimal comma (,). The decimal separator comma is mandatory in the number component.
- A decimal part which consists of zero to n numerics, where n is dependent on the maximum length specified for the component and dependent on the number of characters in the integer part.

To determine the maximum component length, the number of numerics in the integer part, the decimal separator part, and the number of numerics in the decimal part (if present) must be totaled. The character count must be compared against the maximum length parameter which is variable, to ensure component length is not exceeded.

Spurious characters, e.g., punctuation marks or blanks, are not permitted within the integer part or decimal part (if present).

Standard presentation:

If the integer part of the number component <NIP> is defined as Nn, the decimal separator part of the number component is defined as ',' and the decimal part of the number component <NDP> is defined as [Nn], then the number component <NBR> can be defined as <NIP> ',' <NDP>.

Examples:	123,	000100,00
	456,0789	00,0
	3,	000000000000,00
	1,234567	

Maximum decimal digits check:

In some message formats, the number of numerals in the decimal part must be checked against the maximum number allowed for the corresponding currency code or related code word: for the message types where this check is required, the semantic rule 3 or 89 is specified within the message format definition. (see Part V - Table of Messages).

4.7 SUB-6

This function checks if the component in subfield 6 in field 61 is four characters in length such that:

- The first character of the component must be either 'S', 'N' or 'F'.
- If the first character is 'S', the last three characters must be validated by the <MT> function.
- If the first character is 'N' or 'F', the last three characters must consist of alphanumerics.

4.8 SB - LC

The SB-LC function checks the validity of field 22, subfield 2, and field 22C, this field or subfield contains three components, which are defined as follow:

1st component :<SB1><LC1>

2nd component : 4!n

3rd component :<SB2><LC2>

Irrespective of the Message Type being validated, the following checks are performed:

The SB-LC function will check that <SB1><LC1> and <SB2><LC2> appear in fields 22 and 22C in alphabetical order, if this control is negative, the function will issue a T96 error code (see Part IV, Chapter 3).

Depending on the Message Type where fields 22 and 22C is recorded specific checks are performed:

4.8.1 Non-common group messages types (not n92, n95, n96).

1st component :<SB1><LC1>

with	<SB1> = 4!a	representing a bank code (1st four characters of the message originator or receiver, e.g., CHAS in 'CHASUS33')

	<LC1>= 2!c	representing the location code of the message originator or receiver represented in <SB1> (7th and 8th character of the message originator or receiver, e.g., 33 in 'CHASUS33').

2nd component : 4!n

3rd component :<SB2><LC2>

with	<SB2> = 4!a	representing the bank code of the other destination involved in the message exchange
	<LC2>= 2!c	representing the location code of the other destination involved in the message exchange

The components 1 and 3 must contain the respective bank codes and location codes extracted from the originator and receiver mnemonics recorded in the message headers, otherwise the function will issue a T95 error code.

Moreover, in all MTs where the fields 22 and 22C is present, the SB-LC function will check if a common reference mismatch exists between the 2nd component and other fields or subfields . Refer to T22 error code (see Part IV, Chapter 3) for the corresponding MTs and related field where this check is applied. (ex: MT 601 field 32B, subfield 2)

In MTs 360, 361, 362, 364, 365 :

> the 2nd component must be composed of '<u>YYMM</u>' from 'YY<u>YYMM</u>DD' of field 30P.

In other MTs:

> the 4th digit of the 2nd component must be the rightmost non-zero digit of the related field; and the digits 1 through 3 must be the 3 digits to the left of it, 0-filled on the left as needed. The 4 digits must be '0000' if the related field has a value of '0'.

4.8.2 Common group message types n92, n95, n96.

1st component : 4!a 2!c

2nd component : 4!n

3rd component : 4!a 2!c

When the last character of the component 2 consists of a '0', and its preceding character does not consist of a '1', then the entire component 2 must consist of '0000', if this control is negative, the function will issue a T63 error code (see Part IV, Chapter 3).

```
IF 2nd_component = <nnn>"0"
      THEN IF 2nd_component NOT= ("0000" | <nn>"10")
            THEN <errror_T63>
```

The values ’0’ and ’1’ are not permitted in <LC1> and <LC2>, however the value ’0’ is allowed in the second (rightmost) position if the emitter is a Test and Training user. The function will issue a T94 error code.

4.9 Instruction

The function Instruction (or <INSTR>) will validate the content of an instruction and additional information usually found in the field 72, according to the following syntax:

<INSTR>: (’/’1!c’/’[32x])|(’/’2!c’/’[31x])|(’/’3!c’/’[30x])|(’/’4!c’/’[29x])|
(’/’5!c’/’[28x])|(’/’6!c’/’[27x])|(’/’7!c’/’[26x])|(’/’8!c’/’[25x])

Errors found by this validation function will be reported with the usual codes T12, T17, T30, T31, T32, T33, and T34.

4.10 <FLD72> Sender to Receiver (field 72)

The function Sender to Receiver (or <FLD72>) validates the content of the field 72, according to the following syntax:

<FLD72>: <INSTR>
[’CRLF’(<INSTR>|’//’33x)]0-5

Errors found by this validation function will be reported with the usual codes T12, T17, T30, T31, T32, T33, and T34.

In addition to the field 72 structured format validation (i.e. <FLD72>), the ERI (Euro Related Info) validation must be applied, refer to section 4.18 <ERI-FLD72structured> for a complete description of this additional validation.

4.11 VAR-SEQU-1

This function checks the syntax of a sequence of 1 mandatory variable-length subfield plus 1 optional variable-length subfield, separated by two optional slashes (’//’).

This sequence can be found in field 61 (subfields 7 and 8) and field 66A (subfields 4 and 5).

The syntax is expressed as follows:

```
<VAR-SEQU-1> : 16x['//'16x]
```

However, due to the fact that slashes are not considered as component (or subfield) delimiters, since they can be part of the content of these components (or subfields), a specific validation process must be implemented, which is able to handle, with a number of assumptions, the use of these slashes.

This validation process is detailed below.

Let us define the sequence validated by <VAR-SEQU-1> as the set of characters starting from the first character encountered up to the first ’CRLF’ encountered.

```
IF length of the sequence before 'CR' > 34 char
     THEN <error : 'T33'>
ENDIF
IF length of the sequence before 'CR' = 0 char
     THEN <error : 'T17'>
ENDIF

IF the sequence contains one '//'
     THEN
          BEGIN_BLOCK   (It is assumed the 1st man & 2nd opt subfields
                        are both present)
          IF     length of the sequence before '//' > 16 char OR length
                 of the sequence before '//' = 0 char
                 THEN <error : 'T24'>
          ENDIF
          IF the sequence before '//' contains all blanks
               THEN <error : 'T25'>
          ENDIF
          IF the sequence before '//' contains a spurious 'LF'
               THEN <error : 'T31'>
          ENDIF
          IF length of the sequence after '//' > 16 char OR length of
          the sequence after '//' = 0 char
               THEN <error : 'T23'>
          ENDIF
          IF the sequence after '//' contains all blanks
               THEN <error : 'T25'>
          ENDIF
          IF the sequence after '//' contains a spurious 'LF'
               THEN <error : 'T31'>
          ENDIF
          END_BLOCK
     ELSE
          BEGIN_BLOCK (Its assumed only the man subfield
          is present)
          IF length of the sequence > 16 char
               THEN <error : 'T24'>
          ENDIF
          IF the sequence contains all blanks
               THEN <error : 'T25'>
          ENDIF
          IF the sequence contains a spurious 'LF'
               THEN <error : 'T31'>
          ENDIF
          END_BLOCK
ENDIF
```

It is implied in this validation sequence that the text pointer will be adjusted if the '//' sequence is encountered during the validation process.

4.12 VAR-SEQU-2

This function checks the syntax of a sequence of 1 mandatory variable-length subfield plus 1 optional subfield, separated by one optional slash ('/').

This sequence can be found in field 26A (subfields 1 and 2).

The syntax is expressed as follows:

```
<VAR-SEQU-2> : 16x['/'4!x]
```

However, due to the fact that slashes are not considered as component (or subfield) delimiters, since they can be part of the content of these components (or subfields), a specific validation process must be implemented, which is able to handle, with a number of assumptions, the use of these slashes.

This validation process is detailed below.

Let us define the sequence validated by <VAR-SEQU-2> as the set of characters starting from the first character encountered up to the first 'CRLF' encountered.

```
IF length of the sequence > 21 char
     THEN <error : 'T33'>
ENDIF
IF length of the sequence before 'CR' = 0 char
     THEN <error : 'T17'>
ENDIF
IF the sequence contains one '/'
     THEN
          BEGIN_BLOCK (Its assumed the 1st man & 2nd opt subfields are
          both present)
          IF length of the sequence before '/' > 16 char OR length of
          the sequence before '/' = 0 char
               THEN <error : 'T24'>
          ENDIF
          IF the sequence before '/' contains all blanks
               THEN <error : 'T25'>
          ENDIF
          IF the sequence before '/' contains a spurious 'LF'
               THEN <error : 'T31'>
          ENDIF
          IF length of the sequence after '/' NOT= 4 char
               THEN <error : 'T23'>
          ENDIF
          IF the sequence after '/' contains all blanks
               THEN <error : 'T25'>
          ENDIF
          IF the sequence after '/' contains a spurious 'LF'
               THEN <error : 'T31'>
          ENDIF
          END_BLOCK
```

```
      ELSE
            BEGIN_BLOCK (Its assumed only the man subfield is present)
            IF length of the sequence > 16 char
                  THEN <error : 'T24'>
            ENDIF
            IF the sequence contains all blanks
                  THEN <error : 'T25'>
            ENDIF
            IF the sequence contains a spurious 'LF'
                  THEN <error : 'T31'>
            ENDIF
            END_BLOCK
ENDIF
```

It is implied in this validation sequence that the text pointer will be adjusted if the '/' char is encountered during the validation process.

4.13 VAR-SEQU-3

This function checks the syntax of a sequence of 1 mandatory variable-length subfield plus 1 optional variable-length subfield, separated by one optional slash ('/').

This sequence can be found in field 26B (subfields 1 and 2).

The syntax is expressed as follows:

```
<VAR-SEQU-3> : 16x['/'16x]
```

However, due to the fact that slashes are not considered as component (or subfield) delimiters, since they can be part of the content of these components (or subfields), a specific validation process must be implemented, which is able to handle, with a number of assumptions, the use of these slashes.

This validation process is detailed below. Let us define the sequence validated by <VAR-SEQU-3> as the set of characters starting from the first character encountered up to the first 'CRLF' encountered.

```
IF length of the sequence before 'CR' > 33 char
      THEN <error : 'T33'>
ENDIF
IF length of the sequence before 'CR' = 0 char
      THEN <error : 'T17'>
ENDIF
```

```
IF the sequence contains one '/'
     THEN
          BEGIN_BLOCK   It is assumed the 1st man & 2nd opt subfields
                        are both present)
          IF     length of the sequence before '/' > 16 char OR
                 length of the sequence before '/' = 0 char
               THEN <error : 'T24'>
          ENDIF
          IF the sequence before '/' contains all blanks
               THEN <error : 'T25'>
          ENDIF
          IF the sequence before '/' contains a spurious 'LF'
                THEN <error : 'T31'>
          ENDIF
          IF     length of the sequence after '/' > 16 char OR
                 length of the sequence after '/' = 0 char
                THEN <error : 'T23'>
          ENDIF
          IF the sequence after '/' contains all blanks
                THEN <error : 'T25'>
          ENDIF
          IF the sequence after '/' contains a spurious 'LF'
               THEN <error : 'T31'>
          ENDIF
          END_BLOCK
     ELSE

          BEGIN_BLOCK(Its assumed only the man subfield is present)
          IF length of the sequence > 16 char
               THEN <error : 'T24'>
          ENDIF
          IF the sequence contains all blanks
               THEN <error : 'T25'>
          ENDIF
          IF the sequence contains a spurious 'LF'
               THEN <error : 'T31'>
          ENDIF
          END_BLOCK
ENDIF
```

It is implied in this validation sequence that the text pointer will be adjusted if the '/' char is encountered during the validation process.

4.14 VAR-SEQU-4

This function checks the syntax of a sequence of 1 mandatory variable-length subfields and 1 optional variable-length subfield, separated by two optional slashes ('//').

This sequence can be found in field 26C (subfields 5 and 6).

The syntax is expressed as follows:

```
<VAR-SEQU-4> : [4x]['//'8x]
```