# APPLICATION PROGRAM INTERFACE

## INTRODUCTION

This Application Program Interface (API) provides implementation details for adding related hierarchical levels to military shipping label data streams. Adding these hierarchical levels and their relationships enable the user to build a multi-tiered consolidated shipment and to maintain a description of the contents at each hierarchical level. The hierarchical levels and their relationships are in the format of the 'F' data identifier described in Section VI of the Data Application Identifier Standard ANSI MH10.8.2:1995 (R2002) document. This API provides implementation details for both creating and reading data streams with 'F' data identifiers.

The user of this API *creates* a multi-tiered shipment one level at a time. Starting at the lowest level, the user inputs into the API the level being built, and one or more data streams representing shipping data for that level. The output of the API is a single data stream representing all the data combined for that level. This output in turn can be used as a data stream to be included in the next higher hierarchical level of a shipment. Note that shipments do not have to include every hierarchical level, but the user must build the shipment in the proper order of hierarchical levels, starting with the lowest. For example, a shipment might not contain an 'Order' level, but if it does, the 'Order' level must be created before the 'Shipment' level. Table 1 shows the proper order of the hierarchical levels, with 'Shipment' as the highest level and 'Serial #' the lowest. None of the levels are mandatory.

This API also *reads* the multi-level shipment data stream that it previously created. In this instance the user inputs a data stream representing all or part of a consolidated shipment. The API outputs a single data stream containing the data streams that were used to create that shipment level, without the top-level 'F' data identifiers. For example, a user may request the API to read a 'Pack' data stream. The API would output the data streams used to create that Pack, without the Pack 'F' data identifier.

**Table 1 – 'F' Data Identifier Hierarchical Level Codes**

| Level | Identifier | Description |
|---|---|---|
| Shipment | S | Data that applies to the whole shipment, such as bill of lading number, lading quantity, supplier code, etc. |
| Order | O | Data related to the sender's order and the associated receiver's original purchase order. |
| Tare | T | The tare level is used to identify pallets. If there are no identifiable pallets, this level may be omitted. |
| Pack | P | The pack level is used to identify the cartons within which the item is shipped, e.g., label serial numbers. In most cases there will be some sort of packs. |
| Sub-pack | Q | Data related to a grouping of identifiable packages within the pack level. Note that this level is only used when the |

| | | |
|---|---|---|
| | | inner pack has identifiable numbers for each inner pack. |
| Item | I | Stock keeping unit (SKU) identification data. |
| Component | F | Data related to the manufacturer's component |
| Serial # | X | Data related to the manufacturer's serial number |

The 'Shipment' level is unique in that complete shipments can be combined into another shipment. For example, a user in Florida may create an entire shipment, including a 'Shipment' level 'F' data identifier, and send it to New York. A user in California may also create a similar entire shipment and send it to the same person in New York. The New York user may want to combine those 2 shipments into 1 shipment and send it overseas. Allowing the 'Shipment' hierarchical level to be subordinate to itself provides for this capability. This capability is not allowed at any other hierarchical level.

## ASSUMPTIONS AND DEFINITIONS

This document assumes input data streams with the following characteristics:
1. The data stream begins with the Compliance Indicator of '[)>' as the first 3 characters.
2. The data stream ends with the End of Transaction character '$^{E}o_{T}$'. '$^{E}o_{T}$' is counted as a single character in this API.
3. The data element separator '$^{G}_{S}$' may be used throughout the data streams. '$^{G}_{S}$' is counted as a single character in this API.
4. The format trailer separator '$^{R}_{S}$' may be used throughout the data streams. '$^{R}_{S}$' is counted as a single character in this API.

A sequence of characters unique to the API is needed to mark the end of the original data streams that were combined into a single data stream to build the output of the API. A combination of the file separator $^{F}_{S}$ and the plus sign is used for this purpose. '$^{F}_{S}$+' in this API is a 2-character string that indicates the original end of a data stream.

This document defines an 'F' data identifier as being a string or partial data stream of length 8 or 9, beginning with the capital letter 'F' and ending with the data element separator $^{G}_{S}$. See Table 2 for the 'F' data identifier format. All fields are mandatory and are defined as follows:

Hierarchical ID Number – a unique identifying alpha-numeric string of length 2. Answers the question "Who am I?".

Hierarchical Parent ID Number – the Hierarchical ID Number of the parent 'F' data identifier. Answers the question "Who is my parent?".

Hierarchical Child Code – an indicator of the existence of hierarchical levels beneath this 'F' data identifier. 1=yes, 0=no. Answers the question "Am I a parent?".

Hierarchical Level Code – a variable-length field of 1-2 characters indicating the level of the 'F' data identifier. Answers the question "What am I?". Note all of the hierarchical level codes in Table 1 are of length 1, but this format leaves open the possibility for 2-character hierarchical level codes.

**Table 2– ANSI MH10.8.2 Data Identifier "F" Structure**

| Part | String (AN) or Identifier (ID) | Length |
|---|---|---|
| Hierarchical ID Number | AN | 2 |
| Hierarchical Parent ID Number | AN | 2 |
| Hierarchical Child Code | ID | 1 |
| Hierarchical Level Code | ID | 1/2 |

**NOMENCLATURE**

For purposes of this API, logic is written in pseudo PL/SQL code with the following symbol definitions:

'--' comments
'||' string concatenation
'=' equality comparison
'<>' inequality comparison
':=' assignment operator

Standard PL/SQL functions used in this API are listed here with their specifications for clarity:

```
FUNCTION SUBSTR (string_in IN VARCHAR2,
                 start_position_in IN NUMBER,
                 substr_length_in IN NUMBER)  RETURN VARCHAR2;
```

This function returns a substring. 'String_in' is the source string, 'start_position_in' is the starting position of the substring to be returned, 'substr_length_in' is the length of the substring. The last parameter is optional. If it is not present, the function returns all the characters to the end of string_in. Strings in PL/SQL start with position 1.

```
FUNCTION REPLACE (string1 IN VARCHAR2,
                  match_string IN VARCHAR2,
                  replace_string IN VARCHAR2)  RETURN VARCHAR2;
```

This function returns a string in which all occurrences of 'match_string' in 'string1' are replaced with 'replace_string'. The last parameter is optional. If it is not present, the function removes all occurrences of match_string from string1 and returns the resulting string.

FUNCTION LENGTH (string1 IN VARCHAR2) RETURN VARCHAR2;
This function returns the length of string1.

FUNCTION ASCII (single_character IN VARCHAR2) RETURN NUMBER;
This function returns the decimal equivalent of a character in the ASCII character set.

FUNCTION CHR (code_location IN NUMBER) RETURN VARCHAR2;
This function returns the character equivalent of a decimal number in the ASCII character set.

## CONSTANTS

The following constants define the eight 'F' data identifier hierarchical level codes used in this API:

| | |
|---|---|
| Shipment | String := 'S'; |
| Order | String := 'O'; |
| Tare | String := 'T'; |
| Pack | String := 'P'; |
| SubPack | String := 'Q'; |
| Item | String := 'I'; |
| Component | String := 'F'; |
| SerialNumber | String := 'X'; |

## SUBROUTINES

### FUNCTION MAIN

**Purpose:** This function is the main driver of the API. It builds a data stream of one or more shipping labels based on user input.

**Description:** This function accepts the user's request to build or read an 'F' data identifier data stream. If the request is to *build* an 'F' data identifier data stream, this function gets the hierarchical level the user is building, processes one or more input data streams from the user, builds a single data stream with the appropriate 'F' data identifiers added, then returns the new data stream. If the request is to *read* an 'F' data identifier data stream, this function breaks down a single data stream with 'F' data identifiers into the individual data streams from which it originated, and returns those data streams to the user in a single data stream.

**Parameters:**
None.

**Global Variables:**

| | |
|---|---|
| OutputStream | Data Stream --this is the output of the API |
| PrevHL | String representing the most recent hierarchical level processed |
| PrevFStr | String to store the most recent 'F' data identifier processed |

**Local Variables:**

| | |
|---|---|
| InputRequest | String representing user's requested action. For purposes of this API, values will be either 'Build_F_Data' or 'Read_F_Data' |
| InputHL | String representing the 'F' data identifier hierarchical level that the user is currently building. Not used if InputRequest is 'Read_F_Data'. |
| InputLabel | User input data stream. This may be a bar code scan or a data stream from some other source. |

**Return Values:**
Data stream

**Logic:**

```
BEGIN MAIN
   --Ask user whether he or she wants to build a shipment or read a shipment.
   Get InputRequest from user;

   If InputRequest = 'Build_F_Data'
      --User asked to build a shipment label.

      --Ask user what hierarchical level he or she is building.
      Get InputHL from user;

      LOOP
         -- Ask user to input data streams to be included at this level.
         -- Tell user what to input when he or she is finished (this will be the
         -- Done_Indicator).
         Get InputLabel from user;

         EXIT WHEN InputLabel = Done_Indicator;

         Build_Label(InputLabel, InputHL);
      END LOOP;

      OutputStream := OutputStream || '<E>oT'

   Else If InputRequest = 'Read_F_Data'
      --User asked to read a shipment label.
```

```
        -- Ask user to scan label to be read.
        Get InputLabel from user;

        Read_Label(InputLabel);

    End If;

    Return OutputStream;

END MAIN;
```

**Exceptions:**

| | |
|---|---|
| Invalid_Bar_Code | Raise if InputLabel is unreadable. |

**PROCEDURE BUILD_LABEL**

**Purpose:** This procedure builds an 'F' data identifier data stream and stores it in the OutputStream variable.

**Description:** This procedure accepts a data stream and a hierarchical level as input, converts the data stream to a data stream with the appropriate 'F' data identifiers and adds the result to the OutputStream variable. This procedure is called repeatedly from Function Main as long as the user is entering data streams. The last call to this procedure results in a single data stream of all input data streams, including 'F' data identifiers, being stored in the OuputStream variable.

**Parameters:**

| | |
|---|---|
| UserLabel | Data stream input by the user. This may be a bar code scan, a manual entry, or a data stream created by another system. |
| TopHLevel | String representing the 'F' data identifier hierarchical level that the user is currently building. |

**Local Variables:**

| | |
|---|---|
| CurrentHL | String representing current hierarchical level in loop |
| TmpStr | String to store a data element read from UserLabel |
| FStr | String to store a single new 'F' data identifier |
| Prev_Higher_FStr | String to store last 'F' data identifier in OutputStream with next higher hierarchical level |
| F_Id_Num | 2 character string representing 'F' data identifier id number |
| F_Par_Num | 2 character string representing parent id for 'F' data identifier |
| HasKids | Boolean to represent if 'F' data identifier has levels below it<br>0 = no, 1 = yes |

**Return Values:**
None.

**Logic:**

```
BEGIN BUILD_LABEL
    IF OutputStream is empty THEN
        -- this is the first label for OutputStream, so insert the 1st 'F' data identifier

        -- Read characters from UserLabel until '[]>' is read
        -- Raise error if the 1st 3 characters are not '[]>'.
        TmpStr := data element from UserLabel;

        OutputStream := TmpStr || '<R>S' || '06' || '<G>S';
        F_ID_Num := '01';
        F_Par_Num := '00';
        FStr := 'F' || F_ID_Num || F_Par_Num || '0' || TopHLevel || '<G>S';
        OutputStream := OutputStream || FStr;
        PrevFStr := FStr;
        PrevHL := TopHLevel;

    END IF;

    -- Read characters from UserLabel until a '<G>S', or '<E>oT' is read
    TmpStr := next data element from UserLabel ;

    LOOP

        EXIT WHEN SUBSTR(TmpStr, LENGTH(TmpStr), 1) = '<E>oT' OR TmpStr IS NULL;

        IF SUBSTR(TmpStr,1,1) = 'F' AND (LENGTH(TmpStr) = 8 OR LENGTH(TmpStr)
                        = 9) THEN
            -- If TmpStr starts with an 'F' and is of length 8 or 9 (counting the <G>S) then
            -- this is an 'F' data identifier

            HasKids := SUBSTR(TmpStr,6,1);

            IF LENGTH(TmpStr) = 9 THEN
                CurrentHL := SUBSTR(TmpStr,7,2);
            ELSE
                CurrentHL := SUBSTR(TmpStr,7,1);
            END IF;

            IF Hierarchy_Level_Is_Lower(CurrentHL, PrevHL) THEN
                -- the current hierarchical level is lower than the previous one.
```

```
    -- Change character 6 of the last F data in OutputStream to show kids
    -- (set to 1 regardless of its previous value).
    -- Below code uses PL/SQL's REPLACE function
    FStr := SUBSTR(PrevFStr, 1, 5) || '1' || SUBSTR(PrevFStr, 7);
    REPLACE(OutputStream, PrevFStr, FStr);

    F_Id_Num := SUBSTR(PrevFStr,2,2);
    F_Par_Num := F_Id_Num;
    FStr := 'F' || Increment_ID(F_Id_Num) || F_Par_Num || HasKids || CurrentHL ||
                '<GS>';
    OutputStream := OutputStream || FStr;

ELSEIF CurrentHL = PrevHL THEN

    If HasKids = 1 THEN

        F_Id_Num := SUBSTR(PrevFStr,2,2);
        F_Par_Num := SUBSTR(PrevFStr,4,2);

        IF F_Par_Num = '00' THEN
            F_Par_Num := '01';
        END IF;

        FStr := 'F' || Increment_ID(F_Id_Num) || F_Par_Num || HasKids ||
                CurrentHL || '<GS>';
        OutputStream := OutputStream || FStr;

    ELSE
        --this F identifier is the same level as the last one and has no levels
        --beneath it, so it can be eliminated.  Don't add it to OutputStream.
        --Also remove the '<RS>06<GS>' that preceded it from the end of
        --OutputStream.
        OutputStream := SUBSTR(OutputStream,1,LENGTH(OutputStream)-4);
    END IF;

ELSEIF NOT Hierarchy_Level_Is_Lower(CurrentHL, PrevHL) THEN
    -- the current hierarchical level is higher than the previous one

    Prev_Higher_FStr := LAST_F_DATA_WITH_HIGHER_HL(CurrentHL);

    -- Change character 6 of the last F data in OutputStream with the next higher
    -- hierarchical level to show kids (set to 1).
    -- Below code uses PL/SQL's REPLACE function.
    FStr := SUBSTR(Prev_Higher_FStr, 1, 5) || '1' ||
                SUBSTR(Prev_Higher_FStr, 7);
```

```
        REPLACE(OutputStream, Prev_Higher_FStr, FStr);

        F_Par_Num := SUBSTR(Prev_Higher_FStr,2,2);

        --Get the id of the last F data in OutputStream (at whatever level).
        F_Id_Num := SUBSTR(PrevFStr,2,2);

        FStr := 'F' || Increment_ID(F_Id_Num) || F_Par_Num || HasKids ||
                    CurrentHL || '<GS>';

        OutputStream := OutputStream || FStr;

    END IF --  hierarchical level comparison

    PrevFStr := FStr;
    PrevHL := CurrentHL;

    -- read characters from UserLabel until '<GS>', or '<E>oT' is read
    TmpStr := next data element from UserLabel ;

ELSEIF SUBSTR(TmpStr,1,3) = '[)>' AND PrevHL <> TopHLevel THEN
    -- TmpStr starts with a compliance indicator.  Get next data element
    -- from UserLabel until a '<GS>', or '<E>oT' is read.
    TmpStr2 := next data element from UserLabel ;

    -- check to see if it is 'F' data identifier.
    IF SUBSTR(TmpStr2,1,1) = 'F' AND (LENGTH(TmpStr2) = 8 OR
                LENGTH(TmpStr2) = 9) THEN

        -- TmpStr2 is an 'F' data identifier.  Take the compliance indicator off of
        -- TmpStr, append it to OutputStream and put TmpStr2 back into TmpStr
        -- for the next iteration of the loop.
        OutputStream :=  OutputStream || SUBSTR(TmpStr,4);
        TmpStr := TmpStr2;

    ELSE
        -- This is not an 'F' data identifier.  It is a single object to be packed at the
        -- user input hierarchical level.

        -- Strip out the compliance indicator and recombine the 1st 2 data elements.
        TmpStr := SUBSTR(TmpStr,4) || TmpStr2;

        IF TmpStr2 does not end in <E>oT THEN
            -- Read the rest of the label and put all of the data into TmpStr.
            TmpStr2 := next data elements from UserLabel until <E>oT is read;
            TmpStr := TmpStr || TmpStr2;
```

```
            END IF;

            -- Replace the '[E]oT' with '[F]S+'.
            TmpStr := REPLACE(TmpStr, '[E]oT', '[F]S+');

            --Insert the label into OutputStream after the first 'F' top level data identifier
            -- but before any previously scanned 'F' data identifier to preserve the
            -- hierarchical sequence in OutputStream.

            IF LENGTH(TopHLevel) = 1 THEN
                OutputStream := SUBSTR(OutputStream,1,15) || TmpStr ||
                    SUBSTR(OutputStream,16);
            ELSE --TopHLevel is of length 2
                OutputStream := SUBSTR(OutputStream,1,16) || TmpStr ||
                    SUBSTR(OutputStream,17);
            END IF;

            TmpStr := NULL;

        END IF;

    ELSEIF SUBSTR(TmpStr,1,3) = '[]>' AND PrevHL = TopHLevel THEN
        -- Strip out the compliance indicator and continue processing.
        TmpStr := REPLACE(TmpStr, '[]>');

    ELSE
        -- TmpStr is not 'F' identifier data or a data element with
        -- a compliance indicator, so write it out to OutputStream.
        OutputStream := OutputStream || TmpStr;

        -- read characters from UserLabel until a '[G]S', or '[E]oT' is read.
        TmpStr := next data element from UserLabel ;

    END IF; -- TmpStr evaluation

END LOOP;

-- This is the end of UserLabel.
IF TmpStr IS NOT NULL THEN

    -- Raise Incorrect_Format error if TmpStr doesn't end in '[E]oT'.
    -- Strip out any remaining compliance indicator, and replace the '[E]oT' with '[F]S+'.
    TmpStr := REPLACE (TmpStr, '[]>');
    TmpStr := REPLACE (TmpStr, '[E]oT', '[F]S+');
    OutputStream := OutputStream || TmpStr;
END IF;
```

END BUILD_LABEL;

**Exceptions:**

Incorrect_Format Raise if UserLabel does not begin '[)>' and end '$^{E}o_{T}$'.

**FUNCTION HIERARCHY_LEVEL_IS_LOWER**

**Purpose:** This function determines which of two given 'F' data identifier hierarchical levels is the lower.

**Description:** This function accepts two 'F' data identifier hierarchical levels as parameters and determines if the first hierarchical level is lower than the second. Possible hierarchical levels are 'S','O','T','P','Q','I','F','X' and they are ranked from high to low in this order with 'S' being the highest level and 'X' being the lowest. This function returns a Boolean value of TRUE if the first parameter is lower in the ranking than the second parameter. It returns FALSE if the first parameter is greater than or equal to the second parameter.

**Parameters:**

| | |
|---|---|
| HL1 | String representing an 'F' data identifier hierarchical level |
| HL2 | String representing an 'F' data identifier hierarchical level |

**Local Variables:**
None.

**Return Values:**
Boolean

**Logic:**

```
BEGIN HIERARCHY_LEVEL_IS_LOWER
   -- This logic uses the constants defined on page 4.
   IF HL1 = Shipment THEN
      Return FALSE;

   ELSEIF HL1 = Order THEN
      IF HL2 = Shipment THEN
         Return TRUE;
      ELSE
         Return FALSE;
      END IF;

   ELSEIF HL1 = Tare THEN
      IF HL2 IN (Shipment,Order) THEN
```

```
            Return TRUE;
        ELSE
            Return FALSE;
        END IF;

    ELSEIF HL1 = Pack THEN
        IF HL2 IN (Shipment,Order,Tare)  THEN
            Return TRUE;
        ELSE
            Return FALSE;
        END IF;

    ELSEIF HL1 = SubPack THEN
        IF HL2 IN (Shipment,Order,Tare,Pack)  THEN
            Return TRUE;
        ELSE
            Return FALSE;
        END IF;

    ELSEIF HL1 = Item THEN
        IF HL2 IN (Shipment,Order,Tare,Pack,SubPack)  THEN
            Return TRUE;
        ELSE
            Return FALSE;
        END IF;

    ELSEIF HL1 = Component THEN
        IF HL2 IN (Shipment,Order,Tare,Pack,SubPack,Item)  THEN
            Return TRUE;
        ELSE
            Return FALSE;
        END IF;

    ELSEIF HL1 = SerialNumber THEN
        IF HL2 IN (Shipment,Order,Tare,Pack,SubPack,Item,Component)  THEN
            Return TRUE;
        ELSE
            Return FALSE;
        END IF;
    END IF;

END HIERARCHY_LEVEL_IS_LOWER;
```

**Related Information:**
This function is called by BUILD_LABEL.

**FUNCTION INCREMENT_ID**

**Purpose:** This function increments a 2-character 'F' data identifier Id.

**Description**: This function accepts a 2-character 'F' data identifier Id as a parameter and returns the next higher incremented 2-character Id value. This 2-character Id is alphanumeric, making 1,296 unique identifiers possible. Though it is only necessary that each identifier be unique and order is not important in assigning these values, for readability's sake this 2-character Id will be assigned first using numeric combinations, then numeric-alpha combinations, then alpha-numeric combinations, and finally alphabetic combinations. See Table 3 for examples. Note '00' will not be used as an Id. This value is reserved to represent 'no parent' in the 'F' data identifier Parent Id numbers.

**Table 3 – Assignment Order of 'F' Data Identifier Id's**

| Order | Field Type | Field Range | Possible Permutations |
|---|---|---|---|
| 1 | Numeric | 01…99 | 99 |
| 2 | Numeric Alpha | 0A…9Z | 260 |
| 3 | Alpha Numeric | A0…Z9 | 260 |
| 4 | Alphabetic | AA…ZZ | 676 |

**Parameters:**

CurrentId 2-character string representing an 'F' data identifier Id

**Local Variables:**

Column1 1st character of CurrentId
Column2 2nd character of CurrentId
NewId 2-character string representing an 'F' data identifier Id

**Return Values:**

2-character string representing the next 'F' data identifier Id after CurrentId

**Logic:**

```
BEGIN INCREMENT_ID

    IF CurrentId = '99' THEN
        Return '0A'
    ELSEIF CurrentId = '9Z' THEN
        Return 'A0'
    ELSEIF CurrentId = 'Z9' THEN
        Return 'AA'
    ELSE
        Column1 :=  1st character of CurrentId;
```

```
        Column2 :=  2nd character of CurrentId;
        IF Column2 = '9' OR Column2 = 'Z' THEN
            Column1 := NEXT_VALUE(Column1);
        END IF;
        Column2 := NEXT_VALUE(Column2);
        NewId := Column1 || Column2;
        Return NewId;
    END IF;

END INCREMENT_ID;
```

**Related Information:**
This function calls the function NEXT_VALUE detailed below.

**FUNCTION NEXT_VALUE**

**Purpose:** This function increments a single character of the 2-character 'F' data identifier Id.

**Description:** This function accepts a single character of the 2-character 'F' data identifier Id as a parameter and returns the next higher incremented character value.

**Parameters:**

| | |
|---|---|
| IdChar | Character representing a character in the 'F' data identifier Id |

**Local Variables:**

| | |
|---|---|
| DecEquiv | Number to store the decimal equivalent of IdChar |

**Return Values:**
Single Character representing next higher value of IdChar

**Logic:**

```
BEGIN NEXT_VALUE

    IF IdChar = '9' THEN
        Return '0'
    ELSEIF IdChar = 'Z' THEN
        Return 'A'
    ELSE
        --Convert the input parameter ascii character to its decimal equivalent.
        --The below code uses a PL/SQL built-in function called ASCII to do this.
        DecEquiv := ASCII(IdChar);

        --Increment the decimal value by 1
```

```
        DecEquiv := DecEquiv + 1;

        --Convert the incremented value back to a character and return this value.
        --Below code uses the PL/SQL built-in CHR function.
        Return CHR(DecEquiv);

    END IF;

END NEXT_VALUE;
```

**Related Information:**

This function is called only by the INCREMENT_ID function.

**FUNCTION LAST_F_DATA_WITH_HIGHER_HL**

**Purpose:** This function finds and returns the last 'F' data identifier written to the OutputStream that has a higher hierarchical level than the parameter hierarchical level.

**Description:** This function accepts an 'F' data identifier hierarchical level as a parameter and locates the last 'F' data identifier in the OutputStream with the next higher hierarchical level. Note the function searches the OutputStream starting at the end and works backwards. For example: If the input parameter is Pack, this function starts at the end of OutputStream and searches for an 'F' data identifier that has a hierarchical level higher than Pack, such as Shipment, Order, or Tare. This function returns the first 'F' data identifier it finds with a higher level. Since the search started at the end of OutputStream, this would also be the last 'F' data identifier that was written to OutputStream with a higher level. If the parameter 'F' data identifier hierarchical level is the highest one, 'Shipment', the 1st 'F' data identifier in OutputStream is returned. If no 'F' data identifier is found, then empty string is returned.

**Parameters:**

| | |
|---|---|
| TargetHL | String representing a hierarchical level in an 'F' data identifier |

**Local Variables:**

| | |
|---|---|
| TmpHL | String to store a potential hierarchical level |
| TmpChr | String to temporarily store a character of OutputStream |
| Ctr | Loop counter |
| StartSrch | Numeric location where search of OutputStream will begin |

**Return Values:**
String representing an 'F' data identifier

**Logic:**

```
BEGIN LAST_F_DATA_WITH_HIGHER_HL

    IF TargetHL = Shipment THEN
        --Return the first and top-level 'F' data identifier in OutputStream.
        --Raise an error if this isn't also a shipment level 'F' data identifier.

        IF SUBSTR(OutputStream, 15, 1) = 'GS' THEN
            --The top level 'F' data identifier is of length 8
            RETURN  SUBSTR(OutputStream, 8, 8);
        ELSEIF SUBSTR(OutputStream, 16, 1) = 'GS' THEN
            --The top level 'F' data identifier is of length 9
            RETURN  SUBSTR(OutputStream, 8, 9);
        END IF;

    END IF;

    --For TargetHL's lower than Shipment, start at the last possible position
    --of an 'F' data identifier in OutputStream
    StartSrch := LENGTH(OutputStream) – 8;

    -- loop counter starts at StartSrch value and counts backward to 1.
    FOR Ctr IN REVERSE 1 .. StartSrch
    LOOP

        -- get the next character from OutputStream
        TmpChr := SUBSTR(OutputStream, Ctr, 1);

        IF TmpChr = 'F' THEN

            IF SUBSTR(OutputStream, Ctr+7, 1) = 'GS' THEN

                -- This is an 'F' data identifier of length 8.
                -- Get the hierarchical level of this 'F' data identifier.
                TmpHL := SUBSTR(OutputStream, Ctr+6, 1);

                IF Hierarchy_Level_Is_Lower(TargetHL, TmpHL) THEN
                    --The parameter hierarchical level is lower than the hierarchical level
                    -- for this 'F' data identifier, so return its value to the calling program.
                    Return SUBSTR(OutputStream, Ctr, 8);
                END IF;

            ELSEIF SUBSTR(OutputStream, Ctr+8, 1) = 'GS' THEN

                -- This is an 'F' data identifier of length 9.
                -- Get the hierarchical level of this 'F' data identifier.
                TmpHL := SUBSTR(OutputStream, Ctr+6, 2);
```

```
            IF Hierarchy_Level_Is_Lower(TargetHL, TmpHL) THEN
                --The parameter hierarchical level is lower than the hierarchical level
                -- for this 'F' data identifier, so return its value to the calling program.

                Return SUBSTR(OutputStream, Ctr, 9);
            END IF;

        END IF;

    END IF;

END LOOP;

-- no 'F' data identifier was found so return an empty string
RETURN '';

END LAST_F_DATA_WITH_HIGHER_HL;
```

**Related Information:**

This function is called by the BUILD_LABEL function.

**PROCEDURE READ_LABEL**

**Purpose:** This procedure reads an 'F' data identifier data stream and returns the original data streams that were used to create it.

**Description:** This procedure accepts a data stream as a parameter, breaks it down into the original data streams before the 'F' data identifiers were inserted and appends the resulting data streams to the OutputStream variable. At the end of this procedure, the OutputStream variable will consist of consecutive data stream labels with each label beginning with '[)>' and ending with '$^{E}o_{T}$'.

**Parameters:**

| | |
|---|---|
| UserLabel | Data stream representing shipping label with 'F' data identifiers |

**Local Variables:**

| | |
|---|---|
| TmpStr | String to store a data element read from UserLabel |
| PrevLastChr | String to store the last character previously processed |
| CurrLevel | String to store the current hierarchical level |
| HighestLevel | String to store the highest hierarchical level in UserLabel |
| ParentId | String to store a 2-character parent hierarchical level id |

**Return Values:**

None.

**Logic:**

```
BEGIN READ_LABEL
    -- Start the ouput label with the compliance indicator.
    OutputStream := '[)>';

    -- Read characters from UserLabel until '<GS>' is read.
    -- Raise error if this string is not '[)><RS>06<GS>' .
   TmpStr := data element from UserLabel ;

    -- Read characters from UserLabel again until '<GS>' is read.
    -- Raise error if this string is not an 'F' data identifier, but do not add to OutputStream.
    TmpStr := data element from UserLabel ;

    IF LENGTH(TmpStr) = 9 THEN
        HighestLevel := SUBSTR(TmpStr,7,2);
    ELSE
        HighestLevel := SUBSTR(TmpStr,7,1);
    END IF;

    CurrLevel := HighestLevel;
    PrevLastChr := '[)>';

    LOOP

        -- read characters from UserLabel until a '<GS>', '<FS>+', or '<EOT>' is read
        TmpStr := next data element from UserLabel;

        EXIT WHEN TmpStr = '<EOT>';

        IF PrevLastChr = '<EOT>' THEN
            -- Add compliance indicator to start the next label.
            OutputStream := OutputStream || '[)>'
        END IF;

        IF SUBSTR(TmpStr, LENGTH(TmpStr)-1, 2) = '<FS>+' AND CurrLevel = HighestLevel
                        THEN
            -- This marks the end of the original label so substitute '<EOT>' for '<FS>+'.
            -- Note that this is the opposite of what BUILD_LABEL does.
            TmpStr := REPLACE(TmpStr, '<FS>+', '<EOT>');
            OutputStream := OutputStream || TmpStr;
            PrevLastChr := '<EOT>';
```

```
    ELSEIF SUBSTR(TmpStr,1,1) = 'F' AND (LENGTH(TmpStr) = 8 OR
                    LENGTH(TmpStr) = 9) THEN
       -- IF TmpStr starts with an 'F' and is of length 8 or 9 (counting the <G>S) then
       -- this is an 'F' data identifier
       IF LENGTH(TmpStr) = 9 THEN
          -- Get 2-character hierarchical level starting at position 7.
          CurrLevel := SUBSTR(TmpStr,7,2)
       ELSE
          -- Get 1-character hierarchical level starting at position 7.
          CurrLevel := SUBSTR(TmpStr,7,1)
       END IF;

       -- Get the id for this 'F' data identifier's parent.
       ParentId := SUBSTR(Tmpstr,4,2);

       IF ParentId = '01' THEN
          --This 'F' data identifier has the highest 'F' data identifier as a parent.
          --Since that parent has been removed, this 'F'data identifier will now be the
          -- highest hierarchical level for a new data stream.  Change parent id to '00'.

          TmpStr := SUBSTR(TmpStr,1,3) || '00' || SUBSTR(TmpStr,6);

          IF SUBSTR(OutputStream, LENGTH(OutputStream)-5, 6) = '<F>S+<R>S06<G>S'
                    THEN
             -- Take off the '<F>S+<R>S06<G>S' that was written to OutputStream in the last loop
                    iteration.
             OutputStream := SUBSTR(OutputStream, 1, LENGTH(OutputStream) – 6);

             -- Adjust TmpStr to show start of a new data stream.
             TmpStr := '<E>oT' || '[]>' || '<R>s06<G>S' || TmpStr;
          END IF;
       END IF;

       OutputStream := OutputStream || TmpStr;
       PrevLastChr := SUBSTR(TmpStr, LENGTH(TmpStr), 1);

    ELSE
       -- This was original data so write it to the outputstream.
       OutputStream := OutputStream || TmpStr;

       -- Get last character of TmpStr.
       PrevLastChr := SUBSTR(TmpStr, LENGTH(TmpStr), 1);

    END IF; -- TmpStr evaluation

 END LOOP;
```

```
    IF PrevLastChar = '+' THEN
        --Replace the last 'F_S+' with 'E_oT'
        OutputStream := SUBSTR(OutputStream, 1, LENGTH(OutputStream)-2)  || 'E_oT'
    END IF;

END READ_LABEL;
```