EDK II VFR Programming Language Specification

TABLE OF CONTENTS

EDK II VFR Programming Language Specification

DRAFT FOR REVIEW

04/30/2025 09:33:52

Acknowledgements

Revision History

Revision	Revision History	Date
0.90	Preliminary release	December 12, 2007
1.00	First general release	December 21 2007
1.10	Update some syntax and fix some literal issues	February 20, 2008
1.20	Add more defaultstore ID description and fix some literal errors	May 26, 2008
1.30	Add new syntax classguid, suppressifformset, security, formmap to support UEFI 2.3 HII IFR opcodes, update some syntax and fix some literal errors.	Feb 12, 2010
1.40	Add generic Guid opcode, inconsistentif check for EFI_IFR_TIME opcode, and, support question VarStorageId defined by DataType and VarStore name.	May 3 , 2011
1.50	Update efi varstore syntax, add modal, refresh event group and REF5 opcode, and add EFI_HII_REF data type to support UEFI 2.3.1 HII IFR opcodes, corrects Time statement example. Minor editing and	August 31, 2011

	formatting	
1.60	Update syntax for goto, image, questionref and constant value opcodes, correct CALLBACK flag to INTEREACTIVE, correct help string for old syntax date/time example, and add examples for expression opcodes.	December 1, 2011
1.70	Clarify restriction that enum type and struct data filed with more than one dimensions array are not supported.	May 18, 2012
1.80	Add syntax for warningif opcode, update definition for name/value varstore and subtitle opcode, update referenced UEFI spec version info.	Jan 14, 2014
1.90	Correct sample code for catenate/match/cond opcode. Add syntax for match2 opcode. Add sample code to show the buffer type constant value for orderedlist opcode and default opcode.	July 2, 2015
1.91	Convert to Gitbook	April 2017
1.92	#683 VFR Spec: Add union data type and bit fields in VFR Data Struct Definition	Mar 2018

1 INTRODUCTION

1.1 Overview

Internal Forms Representation (IFR) is a binary encoding. This encoding is designed to be comparatively easy to parse and to manipulate programmatically.

To simplify the creation of IFR, a high-level Visual Forms Representation (VFR) language is defined below. Using this language syntax, a compiler can be designed to take an ordinary text file containing VFR as an input, and output IFR for use in a user's program. There are various methods to define the VFR language. This document sets out a method and describes it using BNF-style syntax.

1.2 Assumed Knowledge

A full understanding of the UEFI Specification is assumed throughout this document.

This document is a reference manual for developers adopting the VFR language to create products compliant with current UEFI Specification. It offers enough material to create infrastructure files for the user interface, and to create a driver to export setup- related information to the Human Interface Infrastructure programming interface.

1.3 Related Information

- Unified Extensible Firmware Interface Specification, Unified EFI, Inc., http://www.uefi.org

1.4 Terms

The following terms are used throughout this document to describe varying aspects of input localization:

BNF

BNF is an acronym for "Backus Naur Form". John Backus and Peter Naur introduced for the first time a formal notation to describe the syntax of a given language.

HII

Human Interface Infrastructure. This generally refers to the database that contains string, font, and IFR information along with other pieces that use one of the database components.

IFR

Internal Forms Representation. This is the binary encoding that is used for the representation of user interface pages.

VFR

Visual Forms Representation. This is the source code format that is used by developers to create a user interface with varying pieces of data or questions. This is later compiled into a binary encoding

2 VFR DESCRIPTION IN BNF

This section describes a language for developers to use when writing forms. This language is compiled into an architectural form (i.e., IFR) described in the UEFI Specification. Each section describes a different language terminal or non-terminal.

Note: The text formatting of the VFR statements and examples in this document are consistent with BNF-style conventions rather than with EFI or UEFI conventions.

VFR forms consist of a series of VFR statements. The subsections in this chapter designate the base definitions of the different VFR statements by specifying the keyword "Statement" in the section title, followed by the BNF non-terminal name enclosed in parentheses.

Subsections without the "Statement" keyword in the title are non-terminal definitions referenced by one or more of the VFR statement definitions.

The VFR language uses BNF-style syntax:

- For the BNF-style syntax used here, literal characters and terms are in bold and marked with quotation mark. **Example:** "formset" is composed of literal characters:

  ```
  "formset"
  ```

- Terms are case-sensitive. VFR comments start with "//" and end at the end of the line. **Example:** a comment line:

  ```
  // this is a typical comment marker
  ```

- Optional terms are enclosed in non-bolded braces {}. **Example:** "classguid" definition is optional for formset:

  ```
  { "classguid" "=" classguidDefinition "," }
  ```

- Terms enclosed in parentheses, "()", and followed by an asterisk, "" may be specified in the input VFR zero or more times. *Example: ", vfrStatementStatTag" could appear zero or more times in vfrStatementStatTagList:

  ```
  vfrStatementStatTagList ::=
    vfrStatementStatTag ( "," vfrStatementStatTag )*
  ```

- If the parenthesis is followed by a plus "+" sign, then the term must be present one or more times. **Example:** one or more numbers could be used in ideqvallist expression:

  ```
  ideqvallistExp ::=
    "ideqvallist"
    vfrQuestionDataFieldName "==" ( Number )+
  ```

- Groups of terms are sometimes enclosed in parentheses. The character "|" between the two terms indicates that either term has acceptable syntax. **Example:** either "push" or "pop" is acceptable for vfrPragmaPackType:

```
vfrPragmaPackType ::=
  {
      "show"
    | ( "push" | "pop" ) { "," StringIdentifier } { "," Number }
    | { Number }
  }
```

- A superscript number, "n", followed by a comma "," and another number, "m", such as item{n, m} is used to indicate that the number of occurrences can be from n to m occurrences of the item, inclusive. **Example:** there could be 1 to 8 hexadecimal characters in "Hex8":

```
Hex8 ::= "0x"["0"-"9""A"-"F""a"-"f"]{1,8}
```

2.1 VFR Programming Keywords

The following keywords constitute the working set of commands for the VFR language. An example line of VFR code follows each keyword description to help programmers understand how a sample such code should look.

// (comment marker)

This allows a programmer to leave comments in the VFR file. They have no effect on the IFR binary that is generated. **Example:**

```
// this is a typical comment marker
```

#define

This command is used to assign a meaningful name to a constant, and is very similar in function to the 'C' style. **Example:**

```
#define FORMSET_GUID  {0xA04A27f4,0xDF00,0x4D42,0xB5,0x52,0x39,0x51,0x13,0x02,0x11,0x3D}
```

#include

This command tells the VFR compiler to use the contents of a file as part of the source to compile. **Example:**

```
#include "C:\Source\DriverSampleStrDefs.h"
```

2.2 VFR Program

A complete VFR program takes the following form:

```
vfrProgram ::=
 (
     vfrPragmaPackDefinition
   | vfrDataStructDefinition
 )*
 vfrFormSetDefinition

vfrPragmaPackDefinition ::=
 "#pragma" "pack" "(" vfrPragmaPackType ")"

vfrPragmaPackType ::=
  {
      "show"
    | ( "push" | "pop" ) { "," StringIdentifier } {"," Number }
    | { Number }
  }
```

BEHAVIORS AND RESTRICTIONS

The data structure must be defined before formset statements. The pragma pack number must be the second power of 2.

Example

None.

2.3 VFR Data Struct Definition

```
vfrDataStructDefinition ::=
 { "typedef" } "struct"
 { StringIdentifier }
 "{" vfrDataStructFields "}"
 { StringIdentifier } ";"

vfrDataStructDefinition ::=
 { "typedef" } "union"
 { StringIdentifier }
 "{" vfrDataStructFields "}"
 { StringIdentifier } ";"

vfrDataStructFields ::=
  (
      dataStructField64
    | dataStructField32
    | dataStructField16
    | dataStructField8
    | dataStructFieldBool
    | dataStructFieldString
    | dataStructFieldDate
    | dataStructFieldTime
    | dataStructFieldRef
    | dataStructFieldUser
    | dataStructBitField64
    | dataStructBitField32
    | dataStructBitField16
    | dataStructBitField8
  )*

dataStructField64 ::=
  "UINT64"
  StringIdentifier { "[" Number "]" } ";"

dataStructField32 ::=
  "UINT32"
  StringIdentifier { "[" Number "]" } ";"

dataStructField16 ::=
  "UINT16"
  StringIdentifier { "[" Number "]" } ";"

dataStructField8 ::=
  "UINT8"
  StringIdentifier { "[" Number "]" } ";"

dataStructFieldBool ::=
  "BOOLEAN"
  StringIdentifier { "[" Number "]" } ";"

dataStructFieldString ::=
  "EFI_STRING_ID"
  StringIdentifier { "[" Number "]" } ";"

dataStructFieldDate ::=
  "EFI_HII_DATE"
  StringIdentifier { "[" Number "]" } ";"

dataStructFieldTime ::=
  "EFI_HII_TIME"
  StringIdentifier { "[" Number "]" } ";"

dataStructFieldRef ::=
  "EFI_HII_REF"
  StringIdentifier { "[" Number "]" } ";"
```

```
dataStructFieldUser ::=
  StringIdentifier
  StringIdentifier { "[" Number "]" } ";"

dataStructBitField64 ::=
  "UINT64"
  { StringIdentifier } ":" Number ";"

dataStructBitField32 ::=
  "UINT32"
  { StringIdentifier } ":" Number ";"

dataStructBitField16 ::=
  "UINT16"
  { StringIdentifier } ":" Number ";"

dataStructBitField8 ::=
  "UINT8"
  { StringIdentifier } ":" Number ";"
```

BEHAVIORS AND RESTRICTIONS

The data structure definition is in C-style language. `enum` type is not supported. The keyword of the fields' type must be a user defined data structure or one of these types: `UINT8`, `UINT16`, `UINT32`, `UINT64`, `BOOLEAN`, `EFI_STRING_ID`, `EFI_HII_DATA`, `EFI_HII_TIME EFI_HII_REF`, and at most one-dimensional array is permitted. Note: for the bit field, the number of the bit width could not exceed 32.

Example

```
typedef struct {
  UINT8 mU8;
  UINT16 mU16;
  UINT32 mU32[10];
  UINT64 mU64;
} MyData;

typedef union {
  UINT16   Field16;
  UINT8    Field8;
} MyUnionData;

typedef struct {
  UINT16   Field16;
  UINT8    MyBits1 : 1;
  UINT8    MyBits2 : 3;
  UINT8    MyBits3 : 3;
  UINT16   MyBits4 : 4;
} MyBitsData;
```

Unsupported Example of enum type:

```
typedef enum {
  MyEnumValue1,
  MyEnumValue2,
  MyEnumValueMax
} MyEnum;
```

Unsupported Example of two dimensional arrays in data structure:

```
typedef struct {
  UINT8   mU8;
  UINT32  mU32[10][20];
} MyUnsupportedData;
```

2.4 VFR FormSet Definition

```
vfrFormSetDefinition ::=
  "formset"
  "guid" "=" guidDefinition ","
  "title" "=" getStringId ","
  "help" "=" getStringId ","
  { "classguid" "=" classguidDefinition "," }
  { "class" "=" classDefinition "," }
  { "subclass" "=" subclassDefinition "," }
  vfrFormSetList
  "endformset" ";"

classguidDefinition ::=
  guidDefinition { "|" guidDefinition } { "|" guidDefinition }

classDefinition ::=
  validClassNames ( "|" validClassNames )*

validClassNames ::=
    "NON_DEVICE"
  | "DISK_DEVICE"
  | "VIDEO_DEVICE"
  | "NETWORK_DEVICE"
  | "INPUT_DEVICE"
  | "ONBOARD_DEVICE"
  | "OTHER_DEVICE"
  | Number

subclassDefinition ::=
    "SETUP_APPLICATION"
  | "GENERAL_APPLICATION"
  | "FRONT_PAGE"
  | "SINGLE_USE"
  | Number
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.5 VFR FormSet List Definition

```
vfrFormSetList ::=
  (
      vfrFormDefinition
    | vfrFormMapDefinition
    | vfrStatementImage
    | vfrStatementVarStoreLinear
    | vfrStatementVarStoreEfi
    | vfrStatementVarStoreNameValue
    | vfrStatementDefaultStore
    | vfrStatementDisableIfFormSet
    | vfrStatementSuppressIfFormSet
    | vfrStatementExtension
    )*
```

BEHAVIORS AND RESTRICTIONS

In the formset, there can be more than one form of variable stores and default stores. The formset can control whether to process an individual form by nesting it inside of an `EFI_IFR_DISABLE_IF` expression.

The variable stores and default stores must be defined before being referenced.

Example

None.

2.6 VFR Default Stores Definition

```
vfrStatementDefaultStore ::=
  "defaultstore" StringIdentifier ","
  "prompt" "=" getStringId
  { "," "attribute" "=" Number } ";"
```

BEHAVIORS AND RESTRICTIONS

Note: `attribute` is used to specify the default ID.

It is optional. The Standard Defaults Identifier, `EFI_HII_DEFAULT_CLASS_STANDARD`, is used if the attribute is not defined. The Manufacturing Defaults Identifier, `EFI_HII_DEFAULT_CLASS_MANUFACTURING`, and the Safe Defaults Identifier, `EFI_HII_DEFAULT_CLASS_SAFE`, can be also used if the header file defining them is included.

Example

```
defaultstore MyStandard, prompt = STRING_TOKEN(STR_STANDARD_DEFAULT);
```

2.7 VFR Variable Store Definition

2.7.1 VFR Buffer Store Definition

```
vfrStatementVarStoreLinear ::=
  "varstore"
  (
      StringIdentifier ","
    | "UINT8" ","
    | "UINT16" ","
    | "UINT32" ","
    | "UINT64" ","
    | "EFI_HII_DATE" ","
    | "EFI_HII_TIME" ","
    | "EFI_HII_REF" ","
  )
  { "varid" "=" Number "," }
  "name" "=" StringIdentifier ","
  "guid" "=" guidDefinition ";"
```

BEHAVIORS AND RESTRICTIONS

Note: The `StringIdentifier` following `varstore` is the referred data structure name. The `StringIdentifier` of `name` is the varstore name.

Note: `name` and `guid` are used jointly to specify the variable store.

Example

```
varstore MyData, name = RefName, guid = FORMSET_GUID;
```

2.7.2 VFR EFI Variable Store Definition

```
vfrStatementVarStoreEfi ::=
  "efivarstore"
  (
      StringIdentifier ","
    | "UINT8" ","
    | "UINT16" ","
    | "UINT32" ","
    | "UINT64" ","
    | "EFI_HII_DATE" ","
    | "EFI_HII_TIME" ","
    | "EFI_HII_REF" ","
  )
  { "varid" "=" Number "," }
  "attribute" "=" Number ( "|" Number )* ","
  "name" "=" StringIdentifier ","
  "guid" "=" guidDefinition ";"
```

BEHAVIORS AND RESTRICTIONS

Note: The `StringIdentifier` following `efivarstore` is the referred data structure name. The `StringIdentifier` of `name` is the varstore name.

Note: `name` and `guid` are used jointly to specify the efi variable store.

Example

```
efivarstore EfiDataStructure
  attribute = EFI_VARIABLE_BOOTSERVICE_ACCESS,
  name      = EfiData,
  guid      = GUID;
```

2.7.3 VFR Variable Name Store Definition

```
vfrStatementVarStoreNameValue ::=
  "namevaluevarstore" StringIdentifier ","
  { "varid" "=" Number "," }
  ( "name" "=" getStringId "," )+
  "guid" "=" guidDefinition ";"
```

BEHAVIORS AND RESTRICTIONS

Example

```
namevaluevarstore NameValueVarStore,
  name = STRING_TOKEN(STR_NAMEVALUE_TABLE_ITEM1),
  name = STRING_TOKEN(STR_NAMEVALUE_TABLE_ITEM2),
  name = STRING_TOKEN(STR_NAMEVALUE_TABLE_ITEM3),
  guid = GUID;
```

2.8 VFR FormSet DisableIf Definition

```
vfrStatementDisableIfFormSet ::=
 "disableif" vfrStatementExpression ";"
   vfrFormSetList
 "endif" ";"
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.9 VFR FormSet SuppressIf Definition

```
vfrStatementSuppressIfFormSet ::=
  "suppressif" vfrStatementExpression ";"
    vfrFormSetList
  "endif" ";"
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.10 VFR General Token Definition

2.10.1 GUID Definition

```
Hex8 ::=
  "0x"["0"-"9""A"-"F""a"-"f"]{1,8}

Hex4 ::=
  "0x"["0"-"9""A"-"F""a"-"f"]{1,4}

Hex2 ::=
  "0x"["0"-"9""A"-"F""a"-"f"]{1,2}

guidSubDefinition ::=
  Hex2 "," Hex2 "," Hex2 "," Hex2 ","
  Hex2 "," Hex2 "," Hex2 "," Hex2

guidDefinition ::=
  "{"
  Hex8 "," Hex4 "," Hex4 ","
  (
      "{" guidSubDefinition "}"
    | guidSubDefinition
  )
  "}"
```

BEHAVIORS AND RESTRICTIONS

In practice, the VFR only supports GUIDs in a C-style language structure. It is defined as two 32-bit values, followed by two 16-bit values, followed by eight 1-byte values.

2.10.2 String & String Identifier Definition

```
StringIdentifier ::=
  ["A"-"Z""a"-"z""_"]["A"-"Z""a"-"z""_""0"-"9"]*

getStringId ::=
  "STRING_TOKEN" "(" Number ")"
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.10.3 Number Definition

```
Number ::=
  ( "0x"["0"-"9""A"-"F""a"-"f"]+ ) | ["0"-"9"]+
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.10.4 VFR Statement Header Definition

```
vfrStatementHeader  ::=
```

```
  "prompt" "=" getStringId ","
  "help"   "=" getStringId ","
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.10.5 VFR Question Header Definition

```
vfrQuestionHeader ::=
  { "name" "=" StringIdentifier "," }
  { "varid" "=" vfrStorageVarId "," }
  { "questionid" "=" Number "," }
  vfrStatementHeader

questionheaderFlagsField ::=
    "READ_ONLY"
  | "INTERACTIVE"
  | "RESET_REQUIRED"
  | "OPTIONS_ONLY"
  | "REST_STYLE"

vfrStorageVarId ::=
  ( StringIdentifier "[" Number "]" )
  |
  (
    StringIdentifier
    (
      "." StringIdentifier { "[" Number "]" }
    )*
  )
```

BEHAVIORS AND RESTRICTIONS

Note: `questionid` is used to specify the question ID. If it is not defined, the compiler automatically assigns a unique ID.

Note: `name` is used to specify the reference name, which is optional.

Note: The first `StringIdentifier` defined in `vfrStorageVarId` is the varstore name or the structure name referred by varstore. When the same structure is referred to by more than one varstore statement, only the varstore name can be used here. If it is not defined, this question has no storage.

Example

None.

2.10.6 VFR Constant Value Definition

```
vfrConstantValueField  ::=
    Number
  | "TRUE"
  | "FALSE"
```

```
  | "ONE"
  | "ONES"
  | "ZERO"
  | Number ":" Number ":" Number
  | Number "/" Number "/" Number
  | "STRING_TOKEN" "(" Number ")"
  | { Number ("," Number)* }
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.10.7 VFR Statement Image & Locked Tag Definition

```
vfrImageTag ::=
  "image" "=" "IMAGE_TOKEN" "(" Number ")"

vfrLockedTag ::=
  "locked"

vfrStatementStatTag ::=
  vfrImageTag | vfrLockedTag

vfrStatementStatTagList ::=
  vfrStatementStatTag ( "," vfrStatementStatTag )*
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11 VFR Form Definition

```
vfrFormDefinition ::=
  "form" "formid" "=" Number ","
  "title" "=" getStringId ";"
  (
      vfrStatementImage
    | vfrStatementLocked
    | vfrStatementRules
    | vfrStatementDefault
    | vfrStatementStat
    | vfrStatementQuestions
    | vfrStatementConditional
    | vfrStatementLabel
    | vfrStatementBanner
    | vfrStatementExtension
    | vfrStatementModal
  )*
"endform" ";"
```

BEHAVIORS AND RESTRICTIONS

Note: `formid` must be unique for each form statement in a given formset.

Example

None.

2.11.1 VFR Form Map Definition

```
vfrFormMapDefinition ::=
  "formmap" "formid" "=" Number ","
  (
    "maptitle" "=" getStringId ";"
    "mapguid" "=" guidDefinition ";"
  )*
  (
      vfrStatementImage
    | vfrStatementLocked
    | vfrStatementRules
    | vfrStatementDefault
    | vfrStatementStat
    | vfrStatementQuestions
    | vfrStatementConditional
    | vfrStatementLabel
    | vfrStatementBanner
    | vfrStatementExtension
    | vfrStatementModal
  )*
  "endform" ";"
```

BEHAVIORS AND RESTRICTIONS

Note: `formid` must be unique for each form statement in a given formset.

Example

None.

2.11.2 VFR Image Statement Definition

```
vfrStatementImage ::=
  vfrImageTag ";"
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.3 VFR Locked Statement Definition

```
vfrStatementLocked ::=
  vfrLockedTag ";"
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.4 VFR Rule Statement Definition

```
vfrStatementRules ::=
  "rule" StringIdentifier ","
    vfrStatementExpression
  "endrule" ";"
```

BEHAVIORS AND RESTRICTIONS:

`StringIdentifier` is the name that can be referenced by a question. It should be unique in the formset.

Example

```
rule MyRule, 1 + 2
endrule;
```

2.11.5 VFR Statement Definition

```
vfrStatementStat ::=
    vfrStatementSubTitle
  | vfrStatementStaticText
  | vfrStatementCrossReference
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.5.1 VFR SubTitle Definition

```
vfrStatementSubTitle ::=
  "subtitle"
  "text" "=" getStringId
  { "," "flags" "=" vfrSubtitleFlags }
  (
    { "," vfrStatementStatTagList } ";"
  |
    { "," vfrStatementStatTagList }
    { "," (vfrStatementStat | vfrStatementQuestions)*}
    "endsubtitle" ";"
  )

vfrSubtitleFlags ::=
  subtitleFlagsField ( "|" subtitleFlagsField )* ";"

subtitleFlagsField ::=
    Number
  | "HORIZONTAL"
```

BEHAVIORS AND RESTRICTIONS

Note: `flags` is optional, and the default value is 0.

Example

```
subtitle
  text = STRING_TOKEN(STR_SUBTITLE_TEXT),
  flags = HORIZONTAL;

subtitle
  text = STRING_TOKEN(STR_SUBTITLE_TEXT),

  text
    help  = STRING_TOKEN(STR_TEXT_TEXT),
    text  = STRING_TOKEN(STR_TEXT_TEXT);
    flags = RESET_REQUIRED,
    key   = 0x0001;

endsubtitle;
```

2.11.5.2 VFR Text Definition

```
vfrStatementStaticText ::=
  "text"
  "help" "=" getStringId ","
  "text" "=" getStringId
  { "," "text" "=" getStringId }
  {
    "," "flags" "=" staticTextFlagsField ( "|" staticTextFlagsField )*
    "," "key" "=" Number
  }
  { "," vfrStatementStatTagList } ";"

staticTextFlagsField ::=
    Number
  | questionheaderFlagsField
```

BEHAVIORS AND RESTRICTIONS

Note: `flags` is optional. The default value is 0.

If `EFI_IFR_FLAGS_CALLBACK` is set in `flags` then it will generate an `EFI_IFR_ACTION` op-code. Otherwise, it generates the `EFI_IFR_TEXT` op-code.

The value of `key` will be used as a question ID.

Example

Generates `EFI_IFR_TEXT`:

```
text
  help = STRING_TOKEN(STR_TEXT_TEXT),
  text = STRING_TOKEN(STR_TEXT_TEXT);
```

Generates `EFI_IFR_ACTION`:

```
text
  help  = STRING_TOKEN(STR_TEXT_TEXT),
  text  = STRING_TOKEN(STR_TEXT_TEXT);
  flags = RESET_REQUIRED,
  key   = 0x0001;
```

2.11.5.3 VFR Cross Reference Type Statements Definition

```
vfrStatementCrossReference ::=
    vfrStatementGoto
  | vfrStatementResetButton
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.5.3.1 VFR Goto Statement Definition

```
vfrStatementGoto ::=
  "goto"
  {
    (
      "devicePath"  "=" getStringId ","
      "formsetguid" "=" guidDefinition ","
      "formid"   "=" Number ","
      "question" "=" Number ","
    )
    |
    (
      "formsetguid" "=" guidDefinition ","
      "formid"   "=" Number ","
      "question" "=" Number ","
    )
    |
    (
      "formid"   "=" Number ","
      "question" "="
      (
          StringIdentifier ","
        | Number ","
      )
    )
```

```
    |
    (
      Number ","
    )
  }
  vfrQuestionHeader
  { "," "flags" "=" vfrGotoFlags }
  { "," "key" "=" Number }
  { "," vfrStatementQuestionOptionList }
  ";"

vfrGotoFlags ::=
  gotoFlagsField ( "|" gotoFlagsField )*

gotoFlagsField ::=
    Number
  | questionheaderFlagsField
```

BEHAVIORS AND RESTRICTIONS

The value of `key` is used as a question ID.

Example

Generate `EFI_IFR_REF` without `key`

```
goto 1,
  prompt = STRING_TOKEN(STR_GOTO_PROMPT),
  help   = STRING_TOKEN(STR_GOTO_HELP);
```

Generate `EFI_IFR_REF` with `key`

```
goto 1,
  prompt = STRING_TOKEN(STR_GOTO_PROMPT),
  help   = STRING_TOKEN(STR_GOTO_HELP);
  key    = 0x1234;
```

Generate `EFI_IFR_REF2`

```
goto
  formid   = 1,
  question = QuesttionRef,
  prompt   = STRING_TOKEN(STR_GOTO_PROMPT),
  help     = STRING_TOKEN(STR_GOTO_HELP);
```

Generate `EFI_IFR_REF3`

```
goto
  formsetguid = FORMSET_GUID,
  formid      = 1,
  question    = QuesttionRef,
  prompt      = STRING_TOKEN(STR_GOTO_PROMPT),
  help        = STRING_TOKEN(STR_GOTO_HELP);
```

Generate `EFI_IFR_REF4`

```
goto
  devicepath  = STRING_TOKEN(STR_DEVICE_PATH),
  formsetguid = FORMSET_GUID,
  formid      = 1,
  question    = QuesttionRef,
  prompt      = STRING_TOKEN(STR_GOTO_PROMPT),
```

```
  help       = STRING_TOKEN(STR_GOTO_HELP);
```

Generate `EFI_IFR_REF5` without `varid`

```
goto
  prompt = STRING_TOKEN(STR_GOTO_PROMPT),
  help   = STRING_TOKEN(STR_GOTO_HELP);
```

Generate `EFI_IFR_REF5` with `varid`

```
goto
  varid   = MySTestData.mFieldRef,
  prompt  = STRING_TOKEN(STR_GOTO_PROMPT),
  help    = STRING_TOKEN STR_GOTO_HELP);
  default = FID;QID;GuidValue;STRING_TOKEN(STR_DEVICE_PATH),
  ;
```

Generate `EFI_IFR_REF` with option code

```
goto 1,
  prompt = STRING_TOKEN(STR_GOTO_PROMPT),
  help   = STRING_TOKEN(STR_GOTO_HELP),
  refresh interval = 3
  ;
```

2.11.5.3.2 VFR ResetButton Statement Definition

```
vfrStatementResetButton ::=
  "resetbutton"
  "defaultStore" "=" StringIdentifier ","
  vfrStatementHeader ","
  { vfrStatementStatTagList "," }
  "endresetbutton" ";"
```

BEHAVIORS AND RESTRICTIONS

Note: `defaultStore` should point to the default store defined before.

Example

```
resetbutton
  defaultstore = DefaultStoreRef,
  prompt = STRING_TOKEN(STR_RESET_BUTTON_PROMPT),
  help   = STRING_TOKEN(STR_RESET_BUTTON_HELP),
endresetbutton;
```

2.11.6 VFR Question Type Statements Definition

```
vfrStatementQuestions ::=
    vfrStatementBooleanType
 | vfrStatementDate
 | vfrStatementNumericType
 | vfrStatementStringType
 | vfrStatementOrderedList
 | vfrStatementTime
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.1 VFR Question Tag Definition

```
vfrStatementQuestionTag ::=
    vfrStatementStatTag ","
  | vfrStatementInconsistentIf
  | vfrStatementNoSubmitIf
  | vfrStatementDisableIfQuest
  | vfrStatementRefresh
  | vfrStatementVarstoreDevice
  | vfrStatementExtension
  | vfrStatementRefreshEvent
  | vfrStatementWarningIf
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.1.1 VFR Question Tag InconsistentIf Definition

```
vfrStatementInconsistentIf ::=
  "inconsistentif"
  "prompt" "=" getStringId ","
  vfrStatementExpression
  "endif"
```

BEHAVIORS AND RESTRICTIONS

It can only be used in questions.

Example

```
checkbox
  name       = MyCheckBox,
  varid      = MySTestData.mField1,
  questionid = 0xcb,
  prompt     = STRING_TOKEN(STR_CHECK_BOX_PROMPT),
  help       = STRING_TOKEN(STR_CHECK_BOX_HELP),
  flags      = CHECKBOX_DEFAULT | INTERACTIVE,

  inconsistentif
    prompt = STRING_TOKEN(STR_INCONSISTENT_IF),
    ideqval MySTestData.mField1 == 2007
  endif
endcheckbox;
```

2.11.6.1.2 VFR Question Tag NoSubmitIf Definition

```
vfrStatementNoSubmitIf ::=
  "nosubmitif"
  "prompt" "=" getStringId ","
  vfrStatementExpression
  "endif"
```

BEHAVIORS AND RESTRICTIONS

It can only be used in questions.

Example

```
checkbox
  name       = MyCheckBox,
  varid      = MySTestData.mField1,
  questionid = 0xcb,
  prompt     = STRING_TOKEN(STR_CHECK_BOX_PROMPT),
  help       = STRING_TOKEN(STR_CHECK_BOX_HELP),
  flags      = CHECKBOX_DEFAULT | INTERACTIVE,

  nosubmitif prompt = STRING_TOKEN(STR_NOSUBMIT_IF),
    ideqval MySTestData.mField1 == 2007
  endif
endcheckbox;
```

2.11.6.1.3 VFR Question Tag DisableIf Definition

```
vfrStatementDisableIfQuest ::=
  "disableif" vfrStatementExpression ";"
  vfrStatementQuestionOptionList
  "endif"
```

BEHAVIORS AND RESTRICTIONS

None.

Example

```
checkbox
  name       = MyCheckBox,
  varid      = MySTestData.mField1,
  questionid = 0xcb,
  prompt     = STRING_TOKEN(STR_CHECK_BOX_PROMPT),
  help       = STRING_TOKEN(STR_CHECK_BOX_HELP),
  flags      = CHECKBOX_DEFAULT | INTERACTIVE,

  disableif
    ideqvallist MySTestData.mField1 == 1 3 5 7;
    refresh interval = 1
  endif
endcheckbox;
```

2.11.6.1.4 VFR Question Tag Refresh Definition

```
vfrStatementRefresh ::=
  "refresh" "interval" "=" Number
```

BEHAVIORS AND RESTRICTIONS

It can only be used in questions.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 0xff,
```

```
        refresh interval = 3
endnumeric;
```

2.11.6.1.5 VFR Question Tag VarstoreDevice Definition

```
vfrStatementVarstoreDevice ::=
  "varstoredevice" "=" getStringId ","
```

BEHAVIORS AND RESTRICTIONS

It can only be used in questions.

Example

```
checkbox
  name       = MyCheckBox,
  varid      = MySTestData.mField1,
  questionid = 0xcb,
  prompt     = STRING_TOKEN(STR_CHECK_BOX_PROMPT),
  help       = STRING_TOKEN(STR_CHECK_BOX_HELP),
  flags      = CHECKBOX_DEFAULT | INTERACTIVE,

  varstoredevice = STRING_TOKEN(STR_VARSTOREDEVICE),
endcheckbox;
```

2.11.6.1.6 VFR Question Tag RefreshEvent Definition

```
vfrStatementRefreshEvent ::=
  "refreshguid" "=" guidDefinition ","
```

BEHAVIORS AND RESTRICTIONS

It can only be used in questions.

Example

```
numeric
  varid       = MySTestData.mField2,
  prompt      = STRING_TOKEN(STR_NUMERIC_PROMPT),
  help        = STRING_TOKEN(STR_NUMERIC_HELP),
  flags       = DISPLAY_UINT_HEX,
  minimum     = 0,
  maximum     = 300,
  step        = 0,
  refreshguid = EFI_IFR_REFRESH_ID_OP_GUID,
  default     = 175,
endnumeric;
```

2.11.6.1.7 VFR Question Tag WarningIf Definition vfrStatement

```
WarningIf ::=
  "warningif" "prompt" "=" getStringId "," {"timeout" "=" Number ","}
  vfrStatementExpression
  "endif"
```

BEHAVIORS AND RESTRICTIONS

It can only be used in questions.

Example

```
checkbox
  name       = MyCheckBox,
  varid      = MySTestData.mField1,
  questionid = 0xcb,
  prompt     = STRING_TOKEN(STR_CHECK_BOX_PROMPT),
  help       = STRING_TOKEN(STR_CHECK_BOX_HELP),
  flags      = CHECKBOX_DEFAULT | INTERACTIVE,

  warningif prompt = STRING_TOKEN(STR_INCONSISTENT_IF), timeout = 5,
    ideqval MySTestData.mField1 == 2007
  endif
endcheckbox;
```

2.11.6.2 VFR Question Tag List Definition

```
vfrStatementQuestionTagList ::=
  ( vfrStatementQuestionTag )*
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.3 VFR Question Option Tag Definition

```
vfrStatementQuestionOptionTag ::=
    vfrStatementSuppressIfQuest
  | vfrStatementValue
  | vfrStatementDefault
  | vfrStatementOptions
  | vfrStatementRead
  | vfrStatementWrite
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.3.1 VFR Question SuppressIf Statement Definition

```
vfrStatementSuppressIfQuest ::=
  "suppressif" vfrStatementExpression ";"
    vfrStatementQuestionOptionList
  "endif"
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.3.2 VFR Default Statement Definition

```
vfrStatementDefault ::=
  "default"
  (
    (
        vfrStatementValue ","
```

```
      | "=" vfrConstantValueField ","
    )
    { "defaultstore" "=" StringIdentifier "," }
  )
```

BEHAVIORS AND RESTRICTIONS

It can only be used in a question definition.

Note: `defaultstore` is optional and it points to the default store defined previously. If `defaultstore` is not defined, the `EFIHII_DEFAULT_CLASS_STANDARD` is assigned.

Example

```
default = 1,
default value = 1 + 2,
default = {1,2,3},
```

2.11.6.3.3 VFR Value Statement Definition

```
vfrStatementValue ::=
  "value" "=" vfrStatementExpression ";"
```

BEHAVIORS AND RESTRICTIONS

None.

Example

```
Value = 0;
```

2.11.6.3.4 VFR Option Type Statements Definition

```
vfrStatementOptions ::=
  vfrStatementOneOfOption
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.3.5 VFR OneOf Option Statement Definition

```
vfrStatementOneOfOption ::=
  "option"
  "text"  "=" getStringId ","
  "value" "=" vfrConstantValueField ","
  "flags" "=" vfrOneOfOptionFlags
  ( "," vfrImageTag )*
  ";"

vfrOneOfOptionFlags ::=
  oneofoptionFlagsField ( "|" oneofoptionFlagsField )*

oneofoptionFlagsField ::=
```

```
    Number
  | "OPTION_DEFAULT"
  | "OPTION_DEFAULT_MFG"
  | "INTERACTIVE"
  | "RESET_REQUIRED"
  | "DEFAULT"
  | "REST_STYLE"
```

BEHAVIORS AND RESTRICTIONS

An `option` statement is special; it is used to embellish or describe questions.

These statements can be used to give the possible value and set the default value for questions. In other words, they are not questions, but they influence the questions they embellish. Therefore, the options' `flags` are treated as question `flags` and can accept all values of question `flags`.

Options with the DEFAULT flags can be used to set the default value for questions.

Example

```
option text = STRING_TOKEN(STR_ONE_OF_TEXT), value = 0x2, flags = DEFAULT | RESET_REQUIRED;
```

2.11.6.3.6 VFR Read Statement Definition

```
vfrStatementRead ::=
  "read" vfrStatementExpression ";"
```

BEHAVIORS AND RESTRICTIONS

None.

Example

None

2.11.6.3.7 VFR Write Statement Definition

```
vfrStatementWrite ::=
  "write" vfrStatementExpression ";"
```

BEHAVIORS AND RESTRICTIONS

None.

Example

None

2.11.6.4 VFR Question Tag List Definition

```
vfrStatementQuestionOptionList ::=
  (
      vfrStatementQuestionTag
    | vfrStatementQuestionOptionTag
  )*
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.5 VFR Boolean Type Statement Definition

```
vfrStatementBooleanType ::=
    vfrStatementCheckBox
  | vfrStatementAction
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.5.1 VFR CheckBox Statement Definition

```
vfrStatementCheckBox ::=
  "checkbox"
  vfrQuestionHeader ","
  { "flags" "=" vfrCheckBoxFlags "," }
  { "key" "=" Number "," }
  vfrStatementQuestionOptionList
  "endcheckbox" ";"

vfrCheckBoxFlags ::=
  checkboxFlagsField ( "|" checkboxFlagsField )*

checkboxFlagsField ::=
    Number
  | "CHECKBOX_DEFAULT"
  | "CHECKBOX_DEFAULT_MFG"
  | questionheaderFlagsField
```

BEHAVIORS AND RESTRICTIONS

The value of `key` is used as question ID.

Note: `flags` is optional, and the default value is 0.

Example

```
checkbox
  name    = MyCheckBox,
  varid   = MySTestData.mField1,
  prompt  = STRING_TOKEN(STR_CHECK_BOX_PROMPT),
  help    = STRING_TOKEN(STR_CHECK_BOX_HELP),
  flags   = CHECKBOX_DEFAULT | INTERACTIVE,
  default = TRUE,
endcheckbox;
```

2.11.6.5.2 VFR Action Statement Definition

```
vfrStatementAction ::=
  "action"
  vfrQuestionHeader ","
  { "flags" "=" vfrActionFlags "," }
  "config" "=" getStringId ","
```

```
  vfrStatementQuestionTagList
  "endaction" ";"

vfrActionFlags ::=
  actionFlagsField ( "|" actionFlagsField )*

actionFlagsField ::=
    Number
  | questionheaderFlagsField
```

BEHAVIORS AND RESTRICTIONS

Note: `flags` is optional, and the default value is 0.

Example

```
action
  prompt = STRING_TOKEN(STR_ACTION_PROMPT),
  help   = STRING_TOKEN(STR_ACTION_HELP),
  flags  = INTERACTIVE,
  config = STRING_TOKEN(STR_ACTION_CONFIG),
endaction;
```

2.11.6.6 VFR Numeric Type Statements Definition

```
vfrStatementNumericType ::=
    vfrStatementNumeric
  | vfrStatementOneOf
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.6.1 VFR Numeric Statement Definition

```
vfrStatementNumeric ::=
  "numeric"
  vfrQuestionHeader ,
  { "flags" "=" vfrNumericFlags "," }
  { "key" "=" Number "," }
  vfrSetMinMaxStep
  vfrStatementQuestionOptionList
  "endnumeric" ";"

vfrSetMinMaxStep ::=
  "minimum" "=" Number ","
  "maximum" "=" Number ","
  { "step"  "=" Number "," }

vfrNumericFlags ::=
  numericFlagsField ( "|" numericFlagsField )*

numericFlagsField ::=
    Number
  | "NUMERIC_SIZE_1"
  | "NUMERIC_SIZE_2"
  | "NUMERIC_SIZE_4"
  | "NUMERIC_SIZE_8"
  | "DISPLAY_INT_DEC"
```

```
  | "DISPLAY_UINT_DEC"
  | "DISPLAY_UINT_HEX"
  | questionheaderFlagsField
```

BEHAVIORS AND RESTRICTIONS

Note: `flags` is optional, and the default value partly depends on the size of `varid` defined in `vfrQuestionHeader`.

The default display format is `DISPLAY_UINT_DEC`.

Example

```
numeric
  varid   = STestData.mField2,
  prompt  = STRING_TOKEN(STR_NUMERIC_PROMPT),
  help    = STRING_TOKEN(STR_NUMERIC_HELP),
  flags   = DISPLAY_UINT_HEX,
  minimum = 0,
  maximum = 300,
  step    = 0,
  default = 175,
endnumeric;
```

2.11.6.6.2 VFR OneOf Statement Definition

```
vfrStatementOneOf ::=
  "oneof"
  vfrQuestionHeader,
  { "flags" "=" vfrOneofFlagsField "," }
  { vfrSetMinMaxStep }
  vfrStatementQuestionOptionList
  "endoneof" ";"

vfrOneofFlagsField ::=
  numericFlagsField ( "|" numericFlagsField )*
```

BEHAVIORS AND RESTRICTIONS

Note: `flags` is optional, and the default value partly depends on the size of `varid` defined in `vfrQuestionHeader` syntax.

The flag is defined in the VFR Numeric Statement Definition.

Example

```
oneof
  varid  = STestData.mField3[0],
  prompt = STRING_TOKEN(STR_ONE_OF_PROMPT),
  help   = STRING_TOKEN(STR_ONE_OF_HELP),
  flags  = DISPLAY_UINT_DEC,

  option text = STRING_TOKEN(STR_ONE_OF_TEXT1), value = 0x0, flags = 0;
  option text = STRING_TOKEN(STR_ONE_OF_TEXT2), value = 0x1, flags = 0;
```

```
  option text = STRING_TOKEN(STR_ONE_OF_TEXT3), value = 0x2, flags = DEFAULT;
endoneof;
```

2.11.6.7 VFR String Type Statements Definition

```
vfrStatementStringType ::=
    vfrStatementString
  | vfrStatementPassword
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.6.7.1 VFR String Statement Definition

```
vfrStatementString ::=
  "string"
  vfrQuestionHeader ","
  { "flags" "=" vfrStringFlagsField "," }
  { "key" "=" Number "," }
  "minsize" "=" Number ","
  "maxsize" "=" Number ","
  vfrStatementQuestionOptionList
  "endstring" ";"

vfrStringFlagsField ::=
  stringFlagsField ( "|" stringFlagsField )*

stringFlagsField ::=
    Number
  | "MULTI_LINE"
  | questionheaderFlagsField
```

BEHAVIORS AND RESTRICTIONS

Note: `flags` is optional, and the default value is 0.

Example

```
string
  varid   = STestData.mField1,
  prompt  = STRING_TOKEN(STR_MY_STRING_PROMPT),
  help    = STRING_TOKEN(STR_MY_STRING_HELP),
  flags   = MULTI_LINE,
  minsize = 6,
  maxsize = 0x14,
endstring;
```

2.11.6.7.2 VFR Password Statement Definition

```
vfrStatementPassword ::=
  "password"
  vfrQuestionHeader ,
  { "flags" "=" vfrPasswordFlagsField "," }
  { "key" "=" Number "," }
  "minsize" "=" Number ","
  "maxsize" "=" Number ","
```

```
  vfrStatementQuestionOptionList
  "endpassword" ";"

vfrPasswordFlagsField ::=
  passwordFlagsField ( "|" passwordFlagsField )*

passwordFlagsField ::=
    Number
  | questionheaderFlagsField
```

BEHAVIORS AND RESTRICTIONS

Note: `flags` is optional, and the default value is 0.

The value of `key` is used as a question ID.

Example

```
password
  varid    = STestData.mPrivate.mField2,
  prompt   = STRING_TOKEN(STR_PASSWORD_PROMPT),
  help     = STRING_TOKEN(STR_PASSWORD_HELP),
  minsize = 6,
  maxsize = 20,
endpassword;
```

2.11.6.8 VFR OrderedList Statement Definition

```
vfrStatementOrderedList ::=
  "orderedlist"
  vfrQuestionHeader ","
  { "maxcontainers" "=" Number "," }
  { "flags" "=" vfrOrderedListFlags }
  vfrStatementQuestionOptionList
  "endlist" ";"

vfrOrderedListFlags ::=
   orderedlistFlagsField ( "|" orderedlistFlagsField )*

orderedlistFlagsField ::=
    Number
 | "UNIQUE"
 | "NOEMPTY"
 | questionheaderFlagsField
```

BEHAVIORS AND RESTRICTIONS

Note: `maxcontainers` is optional, and the default value depends on the variable size defined by `varid` in `vfrQuestionHeader`.

Note: `flags` is optional, and the default value is 0.

Example

```
orderedlist
  varid  = STestPriData.mField3,
  prompt = STRING_TOKEN(STR_BOOT_OPTIONS),
  help   = STRING_TOKEN(STR_BOOT_OPTIONS),

  option text = STRING_TOKEN(STR_BOOT_OPTION2), value = 2, flags = RESET_REQUIRED;
  option text = STRING_TOKEN(STR_BOOT_OPTION1), value = 1, flags = RESET_REQUIRED;
  option text = STRING_TOKEN(STR_BOOT_OPTION3), value = 3, flags = RESET_REQUIRED;
  option text = STRING_TOKEN(STR_BOOT_OPTION4), value = 4, flags = RESET_REQUIRED;
  option text = STRING_TOKEN(STR_EMPTY_STRING), value = {1, 2, 3, 4}, flags = DEFAULT;
endlist;
```

2.11.6.9 VFR Date Statement Definition

```
vfrStatementDate ::=
  "date"
  (
    (
      vfrQuestionHeader ,
      { "flags" "=" vfrDateFlags "," }
      vfrStatementQuestionOptionList
    )
    |
    (
      "year" "varid" "=" StringIdentifier "." StringIdentifier ","
      "prompt" "=" getStringId ","
      "help" "=" getStringId ","
      minMaxDateStepDefault

      "month" "varid" "=" StringIdentifier "." StringIdentifier ","
      "prompt" "=" getStringId ","
      "help" "=" getStringId ","
      minMaxDateStepDefault

      "day" "varid" "=" StringIdentifier "." StringIdentifier ","
      "prompt" "=" getStringId ","
      "help" "=" getStringId ","
      minMaxDateStepDefault

      ( vfrStatementInconsistentIf )*
    )
  )
  "enddate" ";"

minMaxDateStepDefault ::=
  "minimum" "=" Number ","
  "maximum" "=" Number ","
  { "step" "=" Number "," }
  { "default" "=" Number "," }

vfrDateFlags ::=
  dateFlagsField ( "|" dateFlagsField )*

dateFlagsField ::=
    Number
  | "YEAR_SUPPRESS"
  | "MONTH_SUPPRESS"
  | "DAY_SUPPRESS"
  | "STORAGE_NORMAL"
  | "STORAGE_TIME"
  | "STORAGE_WAKEUP"
```

BEHAVIORS AND RESTRICTIONS

There are old and new syntax of the VFR Date statement, but they are incompatible. The old VFR Date only uses EFI Date/Time Storage, instead of normal question value storage. The new VFR Date can use either the normal question value storage or EFI Date/Time Storage.

For the new syntax of VFR, `flags` is optional, and the default value is 0 Examples follow.

New syntax of VFR Date:

```
date
  varid   = STestData.mDate,
  prompt  = STRING_TOKEN(STR_DATE_PROMPT),
  help    = STRING_TOKEN(STR_DATE_PROMPT),
  flags   = STORAGE_NORMAL,
  default = 2007/08/26,
enddate;
```

Old syntax of VFR Date:

```
date year varid       = Date.Year,
          prompt      = STRING_TOKEN(STR_DATE_PROMPT),
          help        = STRING_TOKEN(STR_DATE_HELP),
          minimum     = 2003,
          maximum     = 2100,
          step        = 1,
          default     = 2003,

          month varid = Date.Month
          prompt      = STRING_TOKEN(STR_DATE_PROMPT),
          help        = STRING_TOKEN(STR_DATE_HELP),
          minimum     = 1,
          maximum     = 12,
          step        = 1,
          default     = 1,

          day varid   = Date.Day,
          prompt      = STRING_TOKEN(STR_DATE_PROMPT),
          help        = STRING_TOKEN(STR_DATE_HELP),
          minimum     = 1,
          maximum     = 31,
          step        = 0x1,
          default     = 1,
enddate;
```

2.11.6.10 VFR Time Statement Definition

```
vfrStatementTime :
  "time"
  (
    (
      vfrQuestionHeader ","
      { "flags" "=" vfrTimeFlags "," }
      vfrStatementQuestionOptionList
    )
    |
    (
      "hour" "varid" "=" StringIdentifier "." StringIdentifier ","
      "prompt" "=" getStringId ","
      "help" "=" getStringId ","
      minMaxTimeStepDefault

      "minute" "varid" "=" StringIdentifier "." StringIdentifier ","
      "prompt" "=" getStringId ","
      "help" "=" getStringId ","
      minMaxTimeStepDefault

      "second" "varid" "=" StringIdentifier "." StringIdentifier ","
      "prompt" "=" getStringId ","
```

```
      "help" "=" getStringId ","
      minMaxTimeStepDefault

      ( vfrStatementInconsistentIf )*
    )
  )
  "endtime" ";"

minMaxTimeStepDefault ::=
  "minimum"   "=" Number ","
  "maximum"   "=" Number ","
  { "step"    "=" Number "," }
  { "default" "=" Number "," }

vfrTimeFlags ::=
  timeFlagsField ( "|" timeFlagsField )*

timeFlagsField ::=
    Number
  | "HOUR_SUPPRESS"
  | "MINUTE_SUPPRESS"
  | "SECOND_SUPPRESS"
  | "STORAGE_NORMAL"
  | "STORAGE_TIME"
  | "STORAGE_WAKEUP"
```

BEHAVIORS AND RESTRICTIONS:

There are old and new syntax of VFR Time statements. They are incompatible.

- **New:** VFR Time can use either the normal question value storage or EFI Date/Time Storage.
- **Old:** VFR Date can use only EFI Date/Time Storage, instead of normal question value storage.

In the new syntax of VFR, `flags` is optional, and the default value is 0

Example

New Time Syntax:

```
time
      name    = MyTime,
      varid   = STestData.mTime,
      prompt  = STRING_TOKEN(STR_TIME_PROMPT),
      help    = STRING_TOKEN(STR_TIME_PROMPT),
      flags   = STORAGE_NORMAL,
      default = 15:33:33,
endtime;
```

Old Time Syntax:

```
time hour varid = Time.Hour,
     prompt     = STRING_TOKEN(STR_TIME_PROMPT),
     help       = STRING_TOKEN(STR_TIME_HELP),
     minimum    = 0,
     maximum    = 23,
     step       = 1,
     default    = 0,

     minute varid = Time.Minute,
     prompt       = STRING_TOKEN(STR_TIME_PROMPT),
     help         = STRING_TOKEN(STR_TIME_HELP),
     minimum      = 0,
     maximum      = 59,
     step         = 1,
     default      = 0,
```

```
      second varid = Time.Second,
      prompt       = STRING_TOKEN(STR_TIME_PROMPT),
      help         = STRING_TOKEN(STR_TIME_HELP),
      minimum      = 0,
      maximum      = 59,
      step         = 1,
      default      = 0,
endtime;
```

2.11.7 VFR Conditional Type Statements Definition

```
vfrStatementConditional ::=
    vfrStatementDisableIfStat
  | vfrStatementSuppressIfStat
  | vfrStatementGrayOutIfStat
```

Note: There are no BEHAVIORS AND RESTRICTION or an Example for this section.

2.11.7.1 VFR Statement List Definition

```
vfrStatementStatList ::=
    vfrStatementStat
  | vfrStatementQuestions
  | vfrStatementConditional
  | vfrStatementLabel
  | vfrStatementExtension
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.7.2 VFR Statement DisalbeIf Definition

```
vfrStatementDisableIfStat ::=
  "disableif" vfrStatementExpression ";"
  ( vfrStatementStatList )*
  "endif" ";"
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.7.3 VFR Statement SuppressIf Definition

```
vfrStatementSuppressIfStat ::=
  "suppressif" vfrStatementExpression ";"
  ( vfrStatementStatList )*
  "endif" ";"
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.7.4 VFR Statement GrayOutIf Definition

```
vfrStatementGrayOutIfStat ::=
  "grayoutif" vfrStatementExpression ";"
  ( vfrStatementStatList )*
  "endif" ";"
```

Note: There are no BEHAVIORS AND RESTRICTIONS or an Example for this section.

2.11.8 VFR GUID Statement Definition

2.11.8.1 VFR Label Statement Definition

```
vfrStatementLabel ::=
  "label" Number ";"
```

BEHAVIORS AND RESTRICTIONS

It is an extended EFI_IFR_GUID op-code.

It should be paired in the VFR program.

Note: `label` is used to insert op-codes at runtime.

Example

```
label LABEL_START;
label LABEL_END;
```

2.11.8.2 VFR Banner Statement Definition

```
vfrStatementBanner ::=
  "banner" { "," }
  "title" "=" getStringId ","
  (
    (
      "line" Number ","
      "align" ( "left" | "center" | "right" ) ";"
    )
    |
    ( "timeout" "=" Number ";" )
  )
```

BEHAVIORS AND RESTRICTIONS

It is an extended EFI_IFR_GUID op-code.

Example

```
banner
```

```
  title = STRING_TOKEN(STR_BANNER_TITLE),
  line 1,
  align center;
```

2.11.8.3 VFR GUID Extension Definition

```
vfrStatementExtension ::=
  "guidop"
  "guid" "=" guidDefinition
  {
    "," "datatype" "="
    (
        "UINT64" { "[" Number "]" }
      | "UINT32" { "[" Number "]" }
      | "UINT16" { "[" Number "]" }
      | "UINT8" { "[" Number "]" }
      | "BOOLEAN" { "[" Number "]" }
      | "EFI_STRING_ID" { "[" Number "]" }
      | "EFI_HII_DATE" { "[" Number "]" }
      | "EFI_HII_TIME" { "[" Number "]" }
      | "EFI_HII_REF" { "[" Number "]" }
      | StringIdentifier { "[" Number "]" }
    )
    vfrExtensionData
  }
  {
    "," ( vfrStatementExtension )*
    "endguidop"
  }
  ";"

vfrExtensionData ::=
  (
    "," "data" { "[" Number "]" }
    (
      "." StringIdentifier { "[" Number "]" }
    )*
    "=" Number
  )*
```

BEHAVIORS AND RESTRICTIONS

The generic guidop statement is used to specify user extension opcodes. By default, unassigned byte will be zero. The array number in the "data" part cannot be equal to or larger than the one in the "datatype" part. That is, the array index starts from zero.

Example

```
guidop
  guid = GUID,
  datatype = UINT32,
  data = 0x12345678;
```

2.11.9 VFR Modal Statement Definition

```
vfrStatementModal ::=
  modal";"
```

BEHAVIORS AND RESTRICTIONS

It is only used in a form.

Example

```
modal;
```

2.12 VFR Expression Statement Definition

The VFR expression is defined in C-style language.

The syntax of VFR expression is defined as a tree. The positions in the tree are determined according to the priority of the operator (for example: + - * /). At the root of it are the terms of `OR`, followed by the terms of `AND`, because the priority of operator `OR` is lower than the operator `AND`(s). The leaves of the tree are sub- expressions of built-in-functions, unary operators, ternary operators, and constants.

2.12.1 OR

```
vfrStatementExpression ::=
  andTerm ( "OR" andTerm )*
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_OR` op-codes.

2.12.2 AND

```
andTerm ::=
  bitwiseorTerm ( "AND" bitwiseorTerm )*
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_AND` op-codes.

2.12.3 bitwiseor

```
bitwiseorTerm ::=
  bitwiseandTerm ( "|" bitwiseandTerm )*
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_BITWISE_OR` op-codes.

2.12.4 bitwiseand

```
bitwiseandTerm ::=
  equalTerm ( "&" equalTerm )*
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_BITWISE_AND` op-codes.

2.12.5 equal

```
equalTerm ::=
  compareTerm
  (
      "==" compareTerm
    | "!=" compareTerm
```

```
  )*
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_EQUAL` or `EFI_IFR_NOT_EQUAL` op-codes.

2.12.6 compare

```
compareTerm ::=
  shiftTerm
  (
      "<" shiftTerm
    | "<=" shiftTerm
    | ">" shiftTerm
    | ">=" shiftTerm
  )*
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_LESS_THAN EFI_IFR_LESS_EQUAL, EFI_IFR_IFR_GREATER_EQUAL,` or `EFI_IFR_GREATER_THAN` op-codes.

2.12.7 shift

```
shiftTerm ::=
  addMinusTerm
  (
      "<<" addMinusTerm
    | ">>" addMinusTerm
  )*
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_SHIFT_LEFT` or `EFI_IFR_SHIFT_RIGHT` op-codes.

2.12.8 add/minus

```
addMinusTerm ::=
  multdivmodTerm
  (
      "+" multdivmodTerm
    | "-" multdivmodTerm
  )*
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_ADD` or `EFI_IFR_SUBTRACT` op-codes.

2.12.9 multiply/divide/modulo

```
multdivmodTerm ::=
  castTerm
  (
      "*" castTerm
    | "/" castTerm
    | "%" castTerm
  )*
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_MULTIPLY EFI_IFR_MODULO` or `EFI_IFR_DIVIDE` op-codes.

2.12.10 cast terms

```
castTerm ::=
  (
    "("
    (
        "BOOLEAN"
      | "UINT64"
      | "UINT32"
      | "UINT16"
      | "UINT8"
    )
    ")"
  )*
  atomTerm
```

BEHAVIORS AND RESTRICTIONS

The VFR supports the C-style type conversion. The values can be converted into one of the following types: `BOOLEAN`, `UINT64`, `UINT32`, `UINT16`, `UINT8`.

2.12.11 atom terms

```
atomTerm ::=
    vfrExpressionCatenate
  | vfrExpressionMatch
  | vfrExpressionParen
  | vfrExpressionBuildInFunction
  | vfrExpressionConstant
  | vfrExpressionUnaryOp
  | vfrExpressionTernaryOp
  | vfrExpressionMap
  | ( "NOT" atomTerm )
  | vfrExpressionMatch2
```

2.12.11.1 catenate

```
vfrExpressionCatenate ::=
  "catenate"
  "(" vfrStatementExpression "," vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_CATENATE` op-codes.

Example

```
string  varid   = MyData.String,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minsize = 6,
        maxsize = 40,
        inconsistentif prompt = STRING_TOKEN(STR_CHECK),
           pushthis != catenate(stringref(STRING_TOKEN(STR_STRING_RIGHT)),
                                stringref(STRING_TOKEN(STR_STRING_LEFT))),
        endif
```

```
endstring;
```

2.12.11.2 match

```
vfrExpressionMatch ::=
  "match"
  "(" vfrStatementExpression "," vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_MATCH` op-codes.

Example

```
grayoutif  match(stringref(STRING_TOKEN(STR_PATTERN)),
                 stringref(STRING_TOKEN(STR_STRING)))
      numeric varid   = MyData.Data,
              prompt  = STRING_TOKEN(STR_PROMPT),
              help    = STRING_TOKEN(STR_HELP),
              minimum = 0,
              maximum = 243,
      endnumeric;
endif;
```

2.12.11.3 parenthesis

```
vfrExpressionParen ::=
  "(" vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Changes the order of the calculation.

2.12.11.4 build-in functions

```
vfrExpressionBuildInFunction ::=
    dupExp
  | ideqvalExp
  | ideqidExp
  | ideqvallistExp
  | questionref1Exp
  | rulerefExp
  | stringref1Exp
  | pushthisExp
  | securityExp
  | getExp
```

2.12.11.4.1 dup

```
dupExp ::=
  "dup"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_DUP` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 0xf0,
          default value = 2 + dup,
endnumeric;
```

2.12.11.4.2 ideqval

```
ideqvalExp ::=
  "ideqval"
  vfrQuestionDataFieldName "==" Number
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_EQ_ID_VAL` op-codes.

Example

```
grayoutif ideqval MyData.Data1 == 99;
  numeric varid   = MyData.Data,
          prompt  = STRING_TOKEN(STR_PROMPT),
          help    = STRING_TOKEN(STR_HELP),
          minimum = 0,
          maximum = 0xf0,
  endnumeric;
endif;
```

2.12.11.4.3 ideqid

```
ideqidExp ::=
  "ideqid"
  vfrQuestionDataFieldName "==" vfrQuestionDataFieldName
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_EQ_ID_ID` op-codes.

Example

```
grayoutif ideqid MyData.Data2 == MyData.Data3;
  numeric varid   = MyData.Data,
          prompt  = STRING_TOKEN(STR_PROMPT),
          help    = STRING_TOKEN(STR_HELP),
          minimum = 0,
          maximum = 0xf0,
  endnumeric;
endif;
```

2.12.11.4.4 ideqvallist

```
ideqvallistExp ::=
  "ideqvallist"
  vfrQuestionDataFieldName "==" ( Number )+
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_EQ_ID_LIST` op-codes.

```
grayoutif ideqvallist MyData.Data1 == 1 3;
  numeric name    = MyNumeric,
          varid   = MyData.Data,
          prompt  = STRING_TOKEN(STR_PROMPT),
          help    = STRING_TOKEN(STR_HELP),
          minimum = 0,
          maximum = 0xf0,
  endnumeric;
endif;
```

2.12.11.4.5 questionref

```
questionref1Exp ::=
  "questionref"
    "(" StringIdentifier | Number ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_QUESTION_REF1` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 0xf0,
        default value = questionref(MyNumeric),
endnumeric;
```

2.12.11.4.6 ruleref

```
rulerefExp ::=
  "ruleref" "(" StringIdentifier ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_RULE_REF` op-codes.

Example

```
grayoutif ruleref(MyRule) == 1;
  string varid   = MyData.String,
         prompt  = STRING_TOKEN(STR_PROMPT),
         help    = STRING_TOKEN(STR_HELP),
         minsize = 6,
         maxsize = 40,
  endstring;
endif;
```

2.12.11.4.7 stringref

```
stringref1Exp ::=
  "stringref" "(" getStringId ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_STRING_REF1` op-codes.

Example

```
grayoutif stringref(STRING_TOKEN(STR_STRING)) == 1;
  string varid   = MyData.String,
         prompt  = STRING_TOKEN(STR_PROMPT),
         help    = STRING_TOKEN(STR_HELP),
         minsize = 6,
         maxsize = 40,
  endstring;
endif;
```

2.12.11.4.8 pushthis

```
pushthisExp ::=
  "pushthis"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_THIS` op-codes.

Example

```
string  varid   = MyData.String,
        prompt  = STRING_TOKEN(STR_PROMPT), help = STRING_TOKEN(STR_HELP),
        minsize = 6,
        maxsize = 40,
        inconsistentif prompt = STRING_TOKEN(STR_CHECK),
          pushthis != stringref(STRING_TOKEN(STR_STRING))
        endif
endstring;
```

2.12.11.4.9 security

```
securityExp ::=
  "security" "(" guidDefinition ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_SECURITY` op-codes.

Example

```
grayoutif NOT security(EFI_GUID);
    text
      help = STRING_TOKEN(STR_HELP),
      text = STRING_TOKEN(STR_TEXT);
endif;
```

2.12.11.4.10 get

```
getExp ::=
  "get" "(" vfrStorageVarId { "|" "flags" "=" vfrNumericFlags } ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_GET` op-codes.

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        read get(MyData.Data1);
endnumeric;
```

2.12.11.5 constant

```
vfrExpressionConstant ::=
    "TRUE"
  | "FALSE" | "ONE"
  | "ONES"
  | "ZERO"
  | "UNDEFINED"
  | "VERSION"
  | Number
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_TRUE EFI_IFR_FALSE` , `EFI_IFR_ONE EFI_IFR_ONES` , `EFI_IFR_ZERO` , `EFI_IFR_UNDEFINED` , or `EFI_IFR_VERSION` op-codes.

2.12.11.6 unary operators

```
vfrExpressionUnaryOp ::=
    lengthExp
  | bitwisenotExp
  | question23refExp
  | stringref2Exp
  | toboolExp
  | tostringExp
  | unintExp
  | toupperExp
  | tolwerExp
  | setExp
```

2.12.11.6.1 length

```
lengthExp ::=
  "length" "(" vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_LENGTH` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value = length(stringref(STRING_TOKEN(STR_STRING))),
endnumeric;
```

2.12.11.6.2 bitwisenot

```
bitwisenotExp ::=
  "~" "(" vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_BITWISENOT` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value = ~(length(stringref(STRING_TOKEN(STR_STRING)))),
endnumeric;
```

2.12.11.6.3 questionrefval

```
question23refExp ::=
  "questionrefval"
  "("
    {
      DevicePath "=" "STRING_TOKEN" "\(" S:Number "\)" ","
    }
    {
      Uuid "=" guidDefinition[Guid] ","
    }
    vfrStatementExpression
  ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_QUESTION_REF2` or `EFI_IFR_QUESTION_REF3` op-codes.

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value = questionrefval(QuestionID),
endnumeric;
```

2.12.11.6.4 stringrefval

```
stringref2Exp ::=
  "stringrefval" "(" vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_STRING_REF2` op-codes.

Example

```
numeric varid   = MyData.Data,
```

```
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value = length(stringrefval(STR_STRING)),
endnumeric;
```

2.12.11.6.5 boolval

```
toboolExp ::=
  "boolval" "(" vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_TO_BOOLEAN` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255, default value = boolval(12),
endnumeric;
```

2.12.11.6.6 stringval

```
tostringExp ::=
  "stringval" { "format" "=" Number "," }
  "(" vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_TO_STRING` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value = length(stringval(format = 8, 12)),
endnumeric;
```

2.12.11.6.7 unintval

```
unintExp ::=
  "unintval" "(" vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_TO_UINT` op-codes.

Example

```
numeric varid   = MyData.Data,
```

```
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value = unintval(12 * 3),
endnumeric;
```

2.12.11.6.8 toupper

```
toupperExp ::=
  "toupper" "(" vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_TO_UPPER` op-codes.

Example

```
grayoutif length(toupper(stringref(STRING_TOKEN(STR_STRING))))==1;
  string varid   = MyData.String,
         prompt  = STRING_TOKEN(STR_PROMPT),
         help    = STRING_TOKEN(STR_HELP),
         minsize = 6,
         maxsize = 40,
  endstring;
endif;
```

2.12.11.6.9 tolower

```
tolwerExp ::=
  "tolower" "(" vfrStatementExpression ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_TO_LOWER` op-codes.

```
grayoutif length(tolower(stringref(STRING_TOKEN(STR_STRING))))==2;
  string varid   = MyData.String,
         prompt  = STRING_TOKEN(STR_PROMPT),
         help    = STRING_TOKEN(STR_HELP),
         minsize = 6,
         maxsize = 40,
  endstring;
endif;
```

2.12.11.6.10 set

```
setExp ::=
  "set"
  "("
  vfrStorageVarId { "|" "flags" "=" vfrNumericFlags } ","
  vfrStatementExpression
  ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_SET` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        write set(MyData.Data1,10);
endnumeric;
```

2.12.11.7 ternary operators

```
vfrExpressionTernaryOp ::=
    conditionalExp
  | findExp
  | midExp
  | tokenExp
  | spanExp
```

2.12.11.7.1 cond

```
conditionalExp ::=
  "cond"
  "("
  vfrStatementExpression1
  "?"
  vfrStatementExpression2
  ":"
  vfrStatementExpression3
  ")"
```

BEHAVIORS AND RESTRICTIONS

`If (Expr1) then x = Expr3 else Expr2`

Generates `EFI_IFR_CONDITIONAL` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value = cond(2 == 1 ? 5 : 10),
endnumeric;
```

2.12.11.7.2 find

```
findExp ::=
  "find"
  "("
  findFormat ( "|" findFormat )*
  ","
  vfrStatementExpression
  ","
  vfrStatementExpression
  ","
  vfrStatementExpression
  ")"

findFormat ::=
    "SENSITIVE"
  | "INSENSITIVE"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_FIND` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value =
          find(INSENSITIVE, (stringref(STRING_TOKEN(STR_STRING1))),
                            (stringref(STRING_TOKEN(STR_STRING2))), 1),
endnumeric;
```

2.12.11.7.3 mid

```
midExp ::=
  "mid"
  "("
  vfrStatementExpression
  ","
  vfrStatementExpression
  ","
  vfrStatementExpression
  ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_MID` op-codes.

Example

```
grayoutif length(mid(stringref(STRING_TOKEN(STR_STRING)),6,8))==1;
  string varid   = MyData.String,
         prompt  = STRING_TOKEN(STR_PROMPT),
         help    = STRING_TOKEN(STR_HELP),
         minsize = 6,
         maxsize = 40,
  endstring;
endif;
```

2.12.11.7.4 tok

```
tokenExp ::=
  "token"
  "("
  vfrStatementExpression
  ","
  vfrStatementExpression
  ","
  vfrStatementExpression
  ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_TOKEN` op-codes.

Example

```
grayoutif length(token(stringref(STRING_TOKEN(STR_STRING1)),
                 stringref(STRING_TOKEN(STR_STRING2)), 0)) == 2;
  string varid   = MyData.String,
         prompt  = STRING_TOKEN(STR_PROMPT),
         help    = STRING_TOKEN(STR_HELP),
         minsize = 6,
         maxsize = 40,
  endstring;
endif;
```

2.12.11.7.5 span

```
spanExp ::=
  "span"
  "("
  "flags" "=" spanFlags ( "|" spanFlags )*
  ","
  vfrStatementExpression
  ","
  vfrStatementExpression
  ","
  vfrStatementExpression
  ")"

spanFlags ::=
    Number
  | "LAST_NON_MATCH"
  | "FIRST_NON_MATCH"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_SPAN` op-codes.

Example

```
numeric varid = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value = span(flags = LAST_NON_MATCH,
                             stringref(STRING_TOKEN(STR_STRING1)),
                             stringref(STRING_TOKEN(STR_STRING2)),0),
endnumeric;
```

2.12.11.8 map

```
vfrExpressionMap ::=
  "map"
  "("
  vfrStatementExpression
  ":"
  (
    vfrStatementExpression
    ","
    vfrStatementExpression
    ";"
  )*
  ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_MAP` op-codes.

Example

```
numeric varid   = MyData.Data,
        prompt  = STRING_TOKEN(STR_PROMPT),
        help    = STRING_TOKEN(STR_HELP),
        minimum = 0,
        maximum = 255,
        default value = map(pushthis:0,10;1,2;3,5;6,8;),
endnumeric;
```

2.12.11.9 match2

```
vfrExpressionMatch ::=
  "match2"
  "("
  vfrStatementExpressionPattern
  ","
  vfrStatementExpressionString
  ","
  guidDefinition[Guid]
  ")"
```

BEHAVIORS AND RESTRICTIONS

Generates `EFI_IFR_MATCH2` op-codes.

Example

```
grayoutif match2(stringref(STRING_TOKEN(STR_PATTERN)),
                 stringref(STRING_TOKEN(STR_STRING)),
                 PERL_GUID);
    numeric varid   = MyData.Data,
            prompt  = STRING_TOKEN(STR_PROMPT),
            help    = STRING_TOKEN(STR_HELP),
            minimum = 0,
            maximum = 243,
    endnumeric;
endif;
```