920i

Programmable Indicator

irite

Programming Reference

PREPRODUCTION RELEASE VERSION

RICE LAKE WEIGHING SYSTEMS
Industrial Solutions on a Global Scale®

ISO 9001 REGISTERED

67888

Contents

Preproduction Release, December 2001

About This Manual

1.0 Introduction

2.0 Language Syntax

2.1 Lexical Elements

Identifiers

An identifier is a sequence of letters, digits, and underscores. The first character of an identifier must be a letter or an underscore, and the length of an identifier cannot exceed 100 characters. Identifiers are not case-sensitive: `"HELLO"` and `"hello"` are both interpreted as `"HELLO"`.

Examples:

Valid identifiers:

```
Variable12
_underscore
Std_Deviation
```

Not valid identifiers:

`9abc`	First character must be a letter or an underscore.
`ABC DEF`	Space (blank) is not a valid character in an identifier.

Keywords

Keywords are special identifiers that are reserved by the language definition and can only be used as defined by the language. Keywords are listed below:

```
and       array     begin     constant  else      elsif
end       exit      for       function  handler   if
integer   is        loop      mod       not       of
or        procedure program   real      record    return
step      string    then      to        type      var
while
```

Constants

Three types of constants are defined by the language:

Integer Constants: An integer constant is a sequence of decimal digits. The value of an integer constant is limited to the range $0 \ldots 2^{31} - 1$. *Any values outside the allowed range are silently truncated.*

Real Constants: A real constant is an integer constant immediately followed by a decimal point and another integer constant. Real constants conform to the requirements of IEEE-754 for double-precision floating point values.

String Constants: A string constant is a sequence of printable characters delimited by quotation marks (double quotes, `" "`). The maximum length allowed for a string constant is 1000 characters, including the delimiters.

Delimiters

Delimiters include all tokens other than identifiers and keywords, including the arithmetic operators listed below:

```
>=    <=    <>    :=    <>    =     +     –     *     /
.     ,     ;     :     (     )     [     ]     "
```

2.2 Program Structure

A program is delimited by a program header and a matching end statement. The body of a program contains a declarations section, which may be empty, and an optional main code body. The declaration section and the main code body may not both be empty.

```
<program>:
  program IDENTIFIER ';'
    <decl-section>
    <optional-main-body>
  end IDENTIFIER ';'
  ;
<optional-main-body>:
    /* NULL */
  | begin <stmt-list>
  ;
```

The declaration section contains declarations defining global program types, variables, and subprograms. The main code body, if present, is assumed to be the declaration of the program startup event handler. A program startup event is generated when the instrument personality enters operational mode at initial power-up and when exiting setup mode.

Example:

```
program MyProgram;
  KeyCounter : Integer;
  handler AnyKeyPressed;
    begin
      KeyCounter := KeyCounter + 1;
    end;

begin
  KeyCounter := 0
end MyProgram;
```

2.3 Declarations

2.3.1 Type Declarations

Type declarations provide the mechanism for specifying the details of enumeration and aggregate types. The identifier representing the type name must be unique within the scope in which the type declaration appears. All user-defined types must be declared prior to being used.

```
<type-declaration>:
    type IDENTIFIER is <type-definition> ';'
  ;
<type-definition>:
    <record-type-definition>
  | <array-type-definition>
  | <enum-type-definition>
  ;
```

Enumeration Type Definitions

An enumeration type definition defines a finite ordered set of values. Each value, represented by an identifier, must be unique within the scope in which the type definition appears.

```
<enum-type-definition>:
    '(' <identifier-list> ')'
  ;
<identifier-list>:
    IDENTIFIER
```

```
      | <identifier-list> ',' IDENTIFIER
      ;
```

Example:

```
type StopLightColors is (Green, Yellow, Red);
```

Record Type Definitions

A record type definition describes the structure and layout of a record type. Each field declaration describes a named component of the record type. Each component name must be unique within the scope of the record; no two components can have the same name. Enumeration, record and array type definitions are not allowed as the type of a component: only previously defined user- or system-defined type names are allowed.

```
<record-type-definition>:
    record
      <field-declaration-list>
    end record
  ;
<field-declaration-list>:
    <field-declaration>
  | <field declaration-list>
    <field declaration>
  ;
<field-declaration>:
      IDENTIFIER ':' <type> ';'
  ;
```

Example:

```
type MyRecord is
  record
    A : Integer;
    B : Real;
  end record;
```

Database Type Definitions

A database type definition describes a database structure, including an alias used to reference the database.

```
<database-type-definition>:
    database (STRING_CONSTANT)
      <field-declaration-list>
    end database
  ;
<field-declaration-list>:
    <field-declaration>
  | <field declaration-list>
    <field declaration>
  ;
<field-declaration>:
      IDENTIFIER ':' <type> ';'
  ;
```

Example: A database consisting of two fields, an integer field and a real number, could be defined as follows:

```
type MyDB is
    database (dbalias)
      A : Integer
      B : Real
    end database;
```

Array Type Definitions

An array type definition describes a container for an ordered collection of identically typed objects. The container is organized as an array of one or more dimensions. All dimensions begin at index 1.

```
<array-type-definition>:
    array '[' <expr-list> ']' of <type>
  ;
```

Example: A two-dimensional array in which each dimension has an index range of 10 (1…10), for a total of 100 elements could be defined as follows:

```
type MyArray is array [10,10] of Integer;
```

2.3.2 Variable Declarations

A variable declaration creates an object of a particular type. The type specified must be a previously defined user- or system-defined type name. The initial value, if specified, must be type-compatible with the declared object type. All user-defined variables must be declared before being used.

```
<variable-declaration>:
    IDENTIFIER ':' <constant-option> <type>
    <optional-initial-value>
  ;
<constant-option>:
    /* NULL */
  | constant
  ;
<optional-initial-value>:
    /* NULL */
  | := <expr>
  ;
```

Example:

```
MyVariable : StopLightColor;
```

2.3.3 Subprogram Declarations

A subprogram declaration defines the formal parameters, return type, local types and variables, and the executable code of a subprogram.

2.3.4 Handler Declarations

A handler declaration defines a subprogram that is to be installed as an event handler. An event handler does not permit parameters or a return type, and can only be invoked by the event dispatching system.

```
<handler-declaration>:
    handler IDENTIFIER ';'
      <decl-section>
    begin
      <stmt-list>
    end ';'
  ;
```

Example:

```
handler SP1Trip;

I : Integer;

begin
  for I := 1 to 10
  loop
    Writeln (1, "Setpoint Tripped!");
    if I=2 then
```

```
        return;
      endif;
    end loop;
end;
```

2.3.5 Procedure Declarations

A procedure declaration defines a subprogram that can be invoked by other subprograms. A procedure allows parameters but not a return type. A procedure must be declared before it can be referenced; recursion is not supported.

```
<procedure-declaration>:
    procedure IDENTIFIER
    <optional-formal-args> ';'
    <decl-section>
    begin
    <stmt-list>
    end ';'
  ;
<optional-formal-args>:
    /* NULL */
  | <formal-args>
  ;
<formal-args>:
    '(' <arg-list> ')'
  ;
<arg-list>:
    <optional-var-spec>
    <variable-declaration>
  | <arg-list> ';' <optional-var-spec>
    <variable-declaration>
  ;
<optional-var-spec>:
    /* NULL */
  | var
  ;
```

Example:

```
procedure PrintString (S : String);
begin
  Writeln (1, "The String is => ",S);
end;
```

2.3.6 Function Declarations

A function declaration defines a subprogram that can be invoked by other subprograms. A function allows parameters and requires a return type. A function must be declared before it can be referenced; recursion is not supported. A function must return to the point of call using a return-with-value statement.

```
<function-declaration>:
    function IDENTIFIER
    <optoinal-formal-args> ':' <type> ';'
    <decl-section>
    begin
    <stmt-list>
    end ';'
  ;
```

Example:

```
function Sum (A : Integer; B : Integer) : Integer;
begin
  return A + B;
end;
```

2.4 Statements

```
<stmt>:
    <assign-stmt>
  | <call-stmt>
  | <if-stmt>
  | <return-stmt>
  | <loop-stmt>
  | exit-stmt>
  ;
```

2.4.1 Assignment Statement

The assignment statement uses the **:=** delimiter to assign the expression on the right-hand side to the object or component on the left-hand side. The types of the left-hand and right-hand side must be compatible.

```
<assign-stmt>:
    <name> ASSIGN <expr> ';'
  ;
```

Example:

```
MyVariable := Green;
```

2.4.2 Call Statement

The call statement is used to initiate a subprogram invocation. The number and type of any actual parameters are compared against the number and type of the formal parameters that were defined in the subprogram declaration. The number of parameters must match exactly. The types of the actual and formal parameters must also be compatible. Parameter passing is accomplished by copy-in/copy-out.

```
<call-stmt>:
    <name> ';'
;
```

Example:

```
MyProcedure ("Hello");
```

2.4.3 If Statement

```
<if-stmt>:
    if <expr> then
    <stmt-list>
    <optional-elsif-list>
    <optional-else-part>
    end if ';'
  ;
<optional-elsif-list>:
    /* NULL */
  | <elsif-list>
  ;
<elsif-list>:
    elsif <expr> then <stmt-list>
  | <elsif-list>
```

```
      elsif <expr> then <stmt-list>
    ;
  <optional-else-part>:
      /* NULL */
    | else <stmt-list>
    ;
```

Example:

```
      if MyVariable = Green then
         MyVariable := Yellow;
      elsif MyVariable = Yellow then
         MyVariable := Red;
      else
         MyVariable := Green;
      end if;
```

2.4.4 Return Statement

```
  <return-stmt>:
      return <optional-return-value> ';'
    ;
  <optional-return-value>:
      /* NULL */
    | <expr>
    ;
```

Example:

```
      return sin(x);
```

2.4.5 Loop Statement

```
  <loop-stmt>:
      <optional-iteration-clause>
      loop
      <stmt-list>
      end loop ';'
    ;
  <optional-iteration-clause>:
      /* NULL */
    | for <name> ASSIGN <expre> to <expr>
      <optional-step-clause>
    | while <expr>
    ;
  <optional-step-clause>:
      /* NULL */
    | step <expr>
    ;
```

Example:

```
      while I < 10
      loop
        Writeln (1, "I is ", I);
        I := I + 1;
      end loop;
```

2.4.6 Exit Statement

```
<exit-stmt>:
    exit ';'
  ;
```

Example:

```
I := 0;
loop
  Writeln (1, "I is => ", I);
  if I = 10 then
    exit;
  endif;
  I := I = 1;
end loop;
```

2.5 Expressions

2.5.1 Names

Names are bound to program entities by declarations and provide a simple method of entity reference. For example, an integer variable X (declared by `X : Integer`) is referred to by the name X.

Access to a field of a record type variable is accomplished by qualifying the variable name with the field name, using the '.' (dot) operator. Access to an element of an array type variable is accomplished by qualifying the variable name with a subscript list.

```
<name>:
    IDENTIFIER
  | <name> '.' IDENTIFIER
  | <name> '(' <expr-list> ')'
  | <name> '[' <expr-list> ']'
  ;
```

Examples:

```
type MyRecord is
record
  Length : Integer;
  Width : Ineger;
end record;

MyVariable : MyRecord;
MyVariable.Length := 10;
MyVariable.Width := 10;

type MyArray is array [10,10] of Integer;
A : MyArray;
A[5,5] := 100;
```

2.5.2 Discrete Forms

```
<primary>:
    INTEGER_CONSTANT
  | REAL_CONSTANT
  | <name>
  | <string>
  | '(' <expr> ')'
  ;
<string>:
    STRING_CONSTANT
  | <string> STRING_CONSTANT
  ;
```

2.5.3 Unary Operators

```
<primary>:
    '-' <primary>
  | not '(' <primary> ')'
  ;
```

2.5.4 Binary Operators

```
<expr>:
    <logical-expr>
  | <expr> <logical-op> <logical-expr>
  ;
<logical-expr>:
    <term> <bool-op-term>
  ;
<bool-op-term>:
    /* NULL */
  | <boolean-op> <term>
  ;
<term>:
    <factor>
  | <term> <add-op> <factor>
  ;
<factor>:
    <primary>
  | <factor> <mult-op> <primary>
;
```

2.5.5 Logical, Boolean, and Arithmetic Operators

```
<logical-op>:
    and
  | or
  ;

<boolean-op>:
    EQ
  | NE
  | LT
  | GT
  | LE
  | GE
  ;

<mult-op>:
    '*'
  | '/'
  | 'mod'
  ;

<add-op>
    '+'
  | '–'
  ;
```

3.0 Built-in Types and Functions

This section describes the built-in types and functions provided for programming the *920i* indicator. Functions are grouped according to the kinds of operations they support.

3.1 Built-in Types

The following built-in types are used in parameters passed to and from the functions described in this section.

```
type SysCode is (SysOK,
  SysLFTViolation,
  SysOutOfRange,
  SysPermissionDenied,
  SysInvalidScale,
  SysBatchRunning,
  SysBatchNotRunning,
  SysNoTare,
  SysInvalidPort,
  SysInvalidUnits,
  SysInvalidSetpoint,
  SysInvalidRequest,
  SysInvalidMode,
  SysInvalidKey,
  SysInvalidWidget,
  SysInvalidState,
  SysInvalidTimer,
  SysDeviceError,
  SysQFull,
  SysRequestFailed);
type Mode is (GrossMode, NetMode);
type Units is (Primary, Secondary, Tertiary);
type TareType is (NoTare, PushButton, Keyed);
type BatchingMode is (Off, Manual, Auto);
type BatchStatus is (BatchComplete, BatchStopped, BatchRunning, BatchPaused);
type PrintFormat is (GrossFmt, NetFmt,
  AuxFmt,
  TrWInFmt, TrRegFmt, TrWoutFmt,
  SPFmt,
  AccumFmt);
type TimerMode is (TimerOneShot, TimerContinuous);
```

3.1.1 Using SysCode Data

SysCode data can be used to take some action based on whether or not a function completed successfully. For example, the following code checks the SysCode result following a `GetTare` function. If the function completed successfully, the retrieved tare weight is written to Port 1:

```
Scale1 : constant Integer := 1;
Port1 : constant Integer := 1;
SysResult : SysCode;
TareWeight : Real;
…
SysResult:= GetTare (Scale1, Primary, TareWeight);
if SysResult = SysOK then
  Writeln (Port1, "The current tare weight is ", TareWeight)'
end if;
```

3.2 Scale Data Acquisition

NOTE: Unless otherwise stated, when an API with a VAR parameter returns a SysCode value other than *SysOK*, the VAR parameter is not changed.

3.2.1 Weight Acquisition

GetGross

Sets W to the current gross weight value of scale S, in the units specified by U. ***W*** will contain a weight value even if the scale is in programmed overload.

Syntax:

```
function GetGross (S : Integer; U : Units; VAR W : Real) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidUnits	The units specified by ***U*** is not valid.
SysInvalidRequest	The requested value is not available.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
GrossWeight : Real;
…
GetGross (Scale1, Primary, GrossWeight);
Writeln (Port1, "Current gross weight is", GrossWeight);
```

GetNet

Sets ***W*** to the current net weight value of scale S, in the units specified by U. ***W*** will contain a weight value even if the scale is in programmed overload.

Syntax:

```
function GetNet (S : Integer; U : Units; VAR W : Real) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidUnits	The units specified by ***U*** is not valid.
SysInvalidRequest	The requested value is not available.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
NetWeight : Real;
…
GetNet (Scale2, Secondary, NetWeight);
Writeln (Port1, "Current net weight is", NetWeight);
```

GetTare

Sets ***W*** to the tare weight of scale ***S*** in weight units specified by ***U***.

```
function GetTare (S : Integer; U : Units; VAR W : Real) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidUnits	The units specified by ***U*** is not valid.
SysInvalidRequest	The requested value is not available.
SysNoTare	The specified scale has no tare. ***W*** is set to 0.0.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
TareWeight : Real;
…
GetTare (Scale3, Tertiary, TareWeight);
Writeln (Port1, "Current tare weight is ", TareWeight);
```

3.2.2 Tare Manipulation

AcquireTare

Acquires a pushbutton tare from scale ***S***.

Syntax:

```
function AcquireTare (S : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist
SysLFTViolation	The tare operation would violate configured legal-for-trade restrictions for the specified scale. No tare is acquired.
SysOutOfRange	The tare operation would acquire a tare that may cause a display overload. No tare is acquired.
SysPermissionDenied	The tare operation would violate configured tare acquisition restrictions for the specified scale. No tare is acquired.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
AcquireTare (Scale1);
```

SetTare

Sets the tare weight for the specified channel.

Syntax:

```
function SetTare (S : Integer; U : Units; W : Real) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidUnits	The units specified by ***U*** is not valid.
SysLFTViolation	The tare operation would violate configured legal-for-trade restrictions for the specified scale. No tare is acquired.
SysOutOfRange	The tare operation would acquire a tare that may cause a display overload. No tare is acquired.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
DesiredTare : Real;
…
DesiredTare := 1234.5;
SetTare (Scale1, Primary, DesiredTare);
```

GetTareType

Sets T to indicate the type of tare currently on scale S.

Syntax:

```
function GetTareType (S : Integer; VAR T : TareType) : SysCode;
```

TareType values returned:

NoTare	There is no tare value associated with the specified scale.
PushbuttonTare	The current tare was acquired by pushbutton.
Keyed	The current tare was acquired by key entry or by setting the tare.

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist. ***T*** is unchanged.
SysOK	The function completed successfully.

ClearTare

Removes the tare associated with scale ***S*** and sets the tare type associated with the scale to ***NoTare*** .

Syntax:

```
function ClearTare (S : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysNoTare	The scale specified by ***S*** has no tare.

SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
ClearTare (Scale1);
```

3.2.3 Rate of Change

GetROC

Sets ***R*** to the current rate-of-change value of scale S. Syntax:

```
function GetROC (S : Integer; VAR R : Real) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
ROC : Real;
…
GetROC (Scale3, ROC);
Writeln (Port1, "Current ROC is", ROC);
```

3.2.4 Accumulator Operations

GetAccum

Sets ***W*** to the value of the accumulator associated with scale ***S***, in the units specified by ***U***.

Syntax:

```
function GetAccum (S : Integer; U : Units; VAR W ; Real) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidUnits	The units specified by ***U*** is not valid.
SysDeviceError	The scale is reporting an error condition.
SysPermissionDenied	The accumulator is not enabled for the specified scale.
SysOK	The function completed successfully.

Example:

```
AccumValue : Real;
…
GetAccum (Scale1, AccumValue);
```

SetAccum

Sets the value of the accumulator associated with scale ***S*** to weight ***W***, in units specified by ***U***.

Syntax:

```
function SetAccum (S : Integer; U : Units; W : Real) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidUnits	The units specified by ***U*** is not valid.
SysDeviceError	The scale is reporting an error condition.
SysPermissionDenied	The accumulator is not enabled for the specified scale.
SysOK	The function completed successfully.

Example:

```
AccumValue : Real;
…
AccumValue := 110.5
SetAccum (Scale1, Primary, AccumValue);
```

GetAvgAccum

Sets ***W*** to the average accumulator value associated with scale ***S***, in the units specified by ***U***, since the accumulator was last cleared.

Syntax:

```
function GetAvgAccum (S : Integer; U : Units; VAR W ; Real) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidUnits	The units specified by ***U*** is not valid.
SysDeviceError	The scale is reporting an error condition.
SysPermissionDenied	The accumulator is not enabled for the specified scale.
SysOK	The function completed successfully.

Example:

```
AvgAccum : Real;
…
GetAvgAccum (Scale1, AvgAccum);
```

GetAccumCount

Sets ***N*** to the number of accumulations performed for scale ***S*** since its accumulator was last cleared.

Syntax:

```
function GetAccumCount (S : Integer; VAR N ; Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysPermissionDenied	The accumulator is not enabled for the specified scale.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
NumAccums : Integer;
…
GetAccumCount (Scale1, NumAccums);
```

GetAccumDate

Sets ***D*** to the date of the most recent accumulation performed by scale ***S***.

Syntax:

```
function GetAccumDate (S : Integer; VAR D ; String) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysPermissionDenied	The accumulator is not enabled for the specified scale.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
AccumDate : String;
…
GetAccumDate (Scale1, AccumDate);
```

GetAccumTime

Sets ***T*** to the time of the most recent accumulation performed by scale ***S***.

Syntax:

```
function GetAccumTime (S : Integer; VAR T ; String) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysPermissionDenied	The accumulator is not enabled for the specified scale.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
AccumTime : String;
…
GetAccumTime (Scale1, AccumTime);
```

ClearAccum

Sets the value of the accumulator for scale ***S*** to zero.

Syntax:

```
function ClearAccum (S : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysPermissionDenied	The accumulator is not enabled for the specified scale.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
ClearAccum (Scale1);
```

3.2.5 Scale Operations

CurrentScale

Sets ***S*** to the numeric ID of the currently displayed scale.

Syntax:

```
procedure CurrentScale : Integer;
```

Example:

```
ScaleNumber : Integer;
…
CurrentScale (ScaleNumber);
```

SelectScale

Sets scale ***S*** as the current scale.

Syntax:

```
function SelectScale (S : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist. The current scale is not changed
SysOK	The function completed successfully.

Example:

```
SelectScale (Scale1);
```

GetMode

Sets ***M*** to the value representing the current display mode for scale ***S***.

Syntax:

```
function GetMode (S : Integer; VAR M : Mode) : SysCode;
```

Mode values returned:

GrossMode	Scale ***S*** is currently in gross mode.
NetMode	Scale ***S*** is currently in net mode.

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
CurrentMode : Mode;
…
GetMode (Scale1, CurrentMode);
```

Example:

```
TT : TareType;
…
GetTareType (Scale1, TT);
  if TT=KeyedTare then …
```

GetUnits

Sets ***U*** to the value representing the current display units for scale ***S***.

Syntax:

```
function GetUnits (S : Integer; VAR U : Units) : SysCode;
```

Units values returned:

Primary	Primary units are currently displayed on scale ***S***.
Secondary	Secondary units are currently displayed on scale ***S***.

Tertiary	Tertiary units are currently displayed on scale ***S***.

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
CurrentUnits : Units;
…
GetUnits (Scale1, CurrentUnits);
```

GetUnitsString

Sets ***V*** to the text string representing the current display units for scale ***S***.

Syntax:

```
function GetUnitsString (S : Integer; U : Units; VAR V : String) : SysCode;
```

Units values sent:

Primary	Primary units are currently displayed on scale ***S***.
Secondary	Secondary units are currently displayed on scale ***S***.
Tertiary	Tertiary units are currently displayed on scale ***S***.

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidUnits	The units value specified by ***U*** does not exist.
SysOK	The function completed successfully.

Example:

```
CurrentUnitsString : Units;
…
GetUnitsString (Scale1, Primary, CurrentUnitsString);
```

InCOZ

Sets ***V*** to a non-zero value if scale ***S*** is within 0.25 grads of gross zero. If the condition is not met, ***V*** is set to zero.

Syntax:

```
function InCOZ (S : Integer; VAR V : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully

Example:

```
ScaleAtCOZ : Integer;
…
InCOZ (Scale1, ScaleAtCOZ);
```

InMotion

Sets ***V*** to a non-zero value if scale ***S*** is in motion. Otherwise, ***V*** is set to zero.

Syntax:

```
function InMotion (S : Integer; VAR V : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully

Example:

```
ScaleInMotion : Integer;
…
InMotion (Scale1, ScaleInMotion);
```

InRange

Sets ***V*** to zero value if scale ***S*** is in an overload or underload condition. Otherwise, ***V*** is set to a non-zero value.

Syntax:

```
function InRange (S : Integer; VAR V : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist..
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully

Example:

```
ScaleInRange : Integer;
…
InRange (Scale1, ScaleInRange);
```

SetMode

Sets the current display mode on scale ***S*** to ***M***.

Syntax:

```
function SetMode (S : Integer; M : Mode) : SysCode;
```

Mode values sent:

GrossMode	Scale ***S*** is set to gross mode.
NetMode	Scale ***S*** is set to net mode.

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidMode	The mode value ***M*** is not valid.
SysDeviceError	The scale is reporting an error condition. ***M*** is not changed.
SysOK	The function completed successfully.

Example:

```
SetMode (Scale1, Gross);
```

SetUnits

Sets the current display units on scale ***S*** to ***U***.

Syntax:

```
function SetUnits (S : Integer; U : Units) : SysCode;
```

Units values sent:

Primary	Primary units will be displayed on scale ***S***.
Secondary	Secondary units will be displayed on scale ***S***.
Tertiary	Tertiary units will be displayed on scale ***S***.

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidUnits	The units value ***U*** is not valid.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
SetUnits (Scale1, Secondary);
```

ZeroScale

Performs a gross zero scale operation for ***S***.

Syntax:

```
function ZeroScale (S : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist
SysLFTViolation	The zero operation would violate configured legal-for-trade restrictions for the specified scale. No zero is performed.
SysOutOfRange	The zero operation would exceed the configured zeroing limit. No zero is acquired.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
ZeroScale (Scale1);
```

3.2.6 A/D Data

GetFilteredCount

Sets ***C*** to the current filtered A/D count for scale ***S***.

Syntax:

```
function GetFilteredCount (S : Integer; VAR C : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidRequest	The scale specified by ***S*** does not support this operation.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
FilterCount : Integer;
…
GetFilteredCount (1; FilterCount);
```

GetRawCount

Sets ***C*** to the current raw A/D count for scale ***S***.

Syntax:

```
function GetRawCount (S : Integer; VAR C : Integer) : SysCode;
```

SysCode values returned:

SysInvalidScale	The scale specified by ***S*** does not exist.
SysInvalidRequest	The scale specified by ***S*** does not support this operation.
SysDeviceError	The scale is reporting an error condition.
SysOK	The function completed successfully.

Example:

```
RawCount : Integer;
…
GetRawCount (1; RawCount);
```

3.3 System Support

Beep

Causes the internal beeper to emit a 0.1-second beep.

Syntax:

```
procedure Beep;
```

Date$

Returns a string representing the system date contained in ***DT***.

Syntax:

```
function Date$ (DT : DateTime) : String;
```

DisplayIsSuspended

Returns a true (non-zero) value if the display is suspended (using the SuspendDisplay procedure), or a false (zero) value if the display is not suspended.

Syntax:

```
function DisplayIsSuspended : Integer;
```

GetConsecNum

Returns the value of the consecutive number counter.

Syntax:

```
function GetConsecNum : Integer;
```

GetSoftwareVersion

Returns the current software version.

Syntax:

```
function GetSoftwareVersion : String;
```

GetUID

Returns the current unit identifier.

Syntax:

```
function GetUID : Integer;
```

LockKey

Disables the specified front panel key.

Syntax:

```
function LockKey (K : Keys) : SysCode;
```

SysCode values returned:

SysInvalidKey	The key specified is not valid.
SysOK	The function completed successfully.

LockKeypad

Disables operation of the entire front panel keypad.

Syntax:

```
function LockKeypad : SysCode;
```

SysCode values returned:

SysPermissionDenied	
SysOK	The function completed successfully.

ResumeDisplay

Resumes a suspended display.

Syntax:

```
function ResumeDisplay : SysCode;
```

SysCode values returned:

SysOK	The function completed successfully.

SetConsecNum

Sets ***V*** to the value of the consecutive number counter.

Syntax:

```
function SetConsecNum (V : Integer) : SysCode;
```

SysCode values returned:

SysOutOfRange	The value specified is not in the allowed range. The consecutive number is not changed.
SysOK	The function completed successfully.

SetUID

Sets the unit identifier.

Syntax:

```
function SetUID (newid : Integer) : SysCode;
```

SysCode values returned:

SysOutOfRange	The unit identifier specified for ***newid*** is not in the allowed range. The UID is not changed.
SysOK	The function completed successfully.

SuspendDisplay

Suspends the display.

Syntax:

```
function SuspendDisplay : SysCode;
```

SysCode values returned:

SysOK	The function completed successfully.

SystemTime
Returns the current system date and time.

Syntax:
```
function SystemTime : DateTime);
```

Time$
Returns a string representing the system time contained in ***DT***.

Syntax:
```
function Time$ (DT : DateTime) : String;
```

UnockKey
Enables the specified front panel key.

Syntax:
```
function UnlockKey (K : Keys) : SysCode;
```

SysCode values returned:

SysInvalid Key	The key specified is not valid.
SysOK	The function completed successfully.

UnlockKeypad
Enables operation of the entire front panel keypad.

Syntax:
```
function UnlockKeypad : SysCode;
```

SysCode values returned:

SysPermissionDenied	
SysOK	The function completed successfully.

3.4 Setpoints and Batching
NOTE: Unless otherwise stated, when an API with a VAR parameter returns a SysCode value other than *SysOK*, the VAR parameter is not changed.

DisableSP
Disables operation of setpoint ***SP***.

Syntax:
```
function DisableSP (SP : Integer) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysBatchRunning	Setpoint ***SP*** cannot be disabled while a batch is running.
SysInvalidRequest	The setpoint specified by ***SP*** cannot be enabled or disabled.
SysOK	The function completed successfully.

Example:
```
DisableSP (4);
```

EnableSP
Enables operation of setpoint ***SP***.

Syntax:
```
function EnableSP (SP : Integer) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysBatchRunning	Setpoint ***SP*** cannot be enabled while a batch is running.
SysInvalidRequest	The setpoint specified by ***SP*** cannot be enabled or disabled.
SysOK	The function completed successfully.

Example:
```
EnableSP (4);
```

GetBatchingMode
Returns the current batching mode (BATCHNG parameter).

Syntax:

```
function GetBatchingMode : BatchingMode;
```

BatchingMode values returned:

Off	Batching mode is off.
Auto	Batching mode is set to automatic.
Manual	Batching mode is set to manual.

GetBatchStatus

Sets ***S*** to the current batch status.

Syntax:

```
function GetBatchStatus (VAR S : BatchStatus) : SysCode;
```

BatchStatus values returned:

BatchComplete	The batch is complete.
BatchStopped	The batch is stopped.
BatchRunning	A batch routine is in progress.
BatchPaused	The batch is paused.

SysCode values returned:

SysInvalidRequest	The BATCHNG configuration parameter is set to OFF.
SysOK	The function completed successfully.

GetCurrentSP

Sets ***SP*** to the number of the current setpoint.

Syntax:

```
function GetCurrentSP (VAR SP : Integer) : Syscode;
```

SysCode values returned:

SysInvalidRequest	The BATCHNG configuration parameter is set to OFF.
SysBatchNotRunning	No batch routine is running.
SysOK	The function completed successfully.

Example:

```
CurrentSP : Integer;
…
GetCurrentSP (CurrentSP);
Writeln (Port1, "Current setpoint is", CurrentSP);
```

GetSPBand

Sets ***V*** to the current band value (BANDVAL parameter) of the setpoint ***SP***.

Syntax:

```
function GetSPBand (SP : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no hysteresis BANDVAL) parameter.
SysOK	The function completed successfully.

Example:

```
SP7Bandval : Real;
…
GetSPBand (7, SP7BAndval);
Writeln (Port1, "Current Band Value of SP7 is", SP7Bandval);
```

GetSPHyster

Sets ***V*** to the current hysteresis value (HYSTER parameter) of the setpoint ***SP***.

Syntax:

```
function GetSPHyster (SP : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no hysteresis HYSTER) parameter.
SysOK	The function completed successfully.

Example:

```
SP5Hyster : Real;
…
GetSPHyster (5, SP5Hyster);
Writeln (Port1, "Current Hysteresis Value of SP5 is", SP5Hyster);
```

GetSPPreact

Sets ***V*** to the current preact value (PREACT parameter) of the setpoint ***SP***.

Syntax:

```
function GetSPPreact (SP : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no preact (PREACT) parameter.
SysOK	The function completed successfully.

Example:

```
SP2Preval : Real;
…
GetSPPreact (2, SP2Preval);
Writeln (Port1, "Current Preact Value of SP2 is", SP2Preval);
```

GetSPValue

Sets ***V*** to the current value (VALUE parameter) of the setpoint ***SP***.

Syntax:

```
function GetSPValue (SP : Integer; VAR V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no VALUE parameter.
SysOK	The function completed successfully.

Example:

```
SP4Val : Real;
…
GetSPValue (4, SP4Val);
Writeln (Port1, "Current Value of SP4 is", SP4Val);
```

GetSPNSample

For averaging (AVG) setpoints, sets ***N*** to the current number of samples (NSAMPLE parameter) of the setpoint ***SP***.

Syntax:

```
function GetSPNSample (SP : Integer; VAR N : Integer) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no NSAMPLE parameter.
SysOK	The function completed successfully.

Example:

```
SP5NS : Integer;
…
GetSPNSample (5, SP5NS);
Writeln (Port1, "Current NSample Value of SP5 is", SP5NS);
```

GetSPTime

For time of day (TOD) setpoints, sets ***DT*** to the current trip time (TIME parameter) of the setpoint ***SP***.

Syntax:

```
function GetSPTime (SP : Integer; VAR DT : DateTime) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no TIME parameter.
SysOK	The function completed successfully.

Example:

```
SP2TIME : DateTime;
…
GetSPTime (2, SP2TIME);
Writeln (Port1, "Current Trip Time of SP2 is", SP2TIME);
```

GetSPDuration

For time of day (TOD) setpoints, sets ***DT*** to the current trip duration (DURATION parameter) of the setpoint ***SP***.

Syntax:

```
function GetSPDuration (SP : Integer; VAR DT : DateTime) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no DURATION parameter.
SysOK	The function completed successfully.

Example:

```
SP3DUR : DateTime;
…
GetSPTime (3, SP3DUR);
Writeln (Port1, "Current Trip Duration of SP3 is", SP3DUR);
```

GetSPVover

For checkweigh (CHKWEI) setpoints, sets ***V*** to the current overrange value (VOVER parameter) of the setpoint ***SP***.

Syntax:

```
function GetSPVover (SP : Integer; VAR V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no VOVER parameter.
SysOK	The function completed successfully.

Example:

```
SP3VOR : Real;
…
GetSPVover (3, SP3VOR);
Writeln (Port1, "Current Overrange Value of SP3 is", SP3VOR);
```

GetSPVunder

For checkweigh (CHKWEI) setpoints, sets ***V*** to the current overrange value (VUNDER parameter) of the setpoint ***SP***.

Syntax:

```
function GetSPVunder (SP : Integer; VAR V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no VUNDER parameter.
SysOK	The function completed successfully.

Example:

```
SP4VUR : Real;
…
GetSPVunder (4, SP4VUR);
Writeln (Port1, "Current Underrange Value of SP4 is", SP4VUR);
```

PauseBatch

Pauses a currently running batch.

Syntax:

```
function PauseBatch : SysCode;
```

SysCode values returned:

SysBatchRunning	No batch routine is running.
SysOK	The function completed successfully.

ResetBatch

Resets a currently running batch.

Syntax:

```
function ResetBatch : SysCode;
```

SysCode values returned:

SysBatchRunning	No batch routine is running.
SysOK	The function completed successfully.

SetBatchingMode

Sets the batching mode (BATCHNG parameter) to the value specified by ***M***.

BatchingMode values sent:

Off	Batching mode is off.
Auto	Batching mode is set to automatic.
Manual	Batching mode is set to manual.

Syntax:

```
function SetBatchingMode (M : BatchingMode) : SysCode;
```

SysCode values returned:

SysInvalidMode	The batching mode specified by ***M*** is not valid.
SysOK	The function completed successfully.

SetSPBand

Sets the band value (BANDVAL parameter) of setpoint ***SP*** to the value specified by ***V***.

Syntax:

```
function SetSPBand (SP : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no band value (BANDVAL) parameter.
SysBatchRunning	The value cannot be changed because a batch process is currently running.
SysOK	The function completed successfully.

Example:

```
SP7Bandval : Real;
…
SP7Bandval := 10.0
SetSPBand (7, SP7Bandval);
```

SetSPHyster

Sets the hysteresis value (HYSTER parameter) of setpoint ***SP*** to the value specified by ***V***.

Syntax:

```
function SetSPHyster (SP : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no hysteresis (HYSTER) parameter.
SysBatchRunning	The value cannot be changed because a batch process is currently running.
SysOK	The function completed successfully.

Example:

```
SP5Hyster : Real;
…
SP5Hyster := 15.0;
SetSPHyster (5, SP5Hyster);
```

SetSPPreact

Sets the preact value (PREACT parameter) of setpoint ***SP*** to the value specified by ***V***.

Syntax:

```
function SetSPPreact (SP : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.

SysInvalidRequest	The setpoint specified by ***SP*** has no preact (PREACT) parameter.
SysBatchRunning	The value cannot be changed because a batch process is currently running.
SysOK	The function completed successfully.

Example:

```
SP2PreVal : Real;
…
SP2PreVal := 30.0;
SetSPPreact (2, SP2PreVal);
```

SetSPValue

Sets the value (VALUE parameter) of setpoint ***SP*** to the value specified by ***V***.

Syntax:

```
function SetSPValue (SP : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no VALUE parameter.
SysBatchRunning	The value cannot be changed because a batch process is currently running.
SysOutOfRange	The value specified for ***V*** is not in the allowed range for setpoint ***SP***.
SysOK	The function completed successfully.

Example:

```
SP4Val : Real;
…
SP4Val := 350.0;
SetSPValue (4, SP4Val);
```

SetSPNSample

For averaging (AVG) setpoints, sets the number of samples (NSAMPLE parameter) of setpoint ***SP*** to the value specified by ***N***.

Syntax:

```
function SetSPNSample (SP : Integer; N : Integer) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no NSAMPLE parameter.
SysBatchRunning	The value cannot be changed because a batch process is currently running.
SysOutOfRange	The value specified for ***N*** is not in the allowed range for setpoint ***SP***.
SysOK	The function completed successfully.

Example:

```
SP5NS : Integer;
…
SP5NS := 10
SetSPNSample (5, SP5NS);
```

SetSPVover

For checkweigh (CHKWEI) setpoints, sets the overrange value (VOVER parameter) of setpoint ***SP*** to the value specified by ***V***.

Syntax:

```
function SetSPVover (SP : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no VOVER parameter.
SysOK	The function completed successfully.

Example:

```
SP3VOR : Real;
…
SP3VOR := 35.5
SetSPVover (3, SP3VOR);
```

SetSPVover

For checkweigh (CHKWEI) setpoints, sets the underrange value (VUNDER parameter) of setpoint ***SP*** to the value specified by ***V***.

Syntax:

```
function SetSPVunder (SP : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no VUNDER parameter.
SysOK	The function completed successfully.

Example:

```
SP4VUR : Real;
…
SP4VUR := 26.4
SetSPVunder (4, SP4VUR);
```

SetSPTime

For time of day (TOD) setpoints, sets the trip time (TIME parameter) of setpoint ***SP*** to the value specified by ***DT***.

Syntax:

```
function SetSPTime (SP : Integer; DT : DateTime) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no TIME parameter.
SysBatchRunning	The value cannot be changed because a batch process is currently running.
SysOutOfRange	The value specified for ***DT*** is not in the allowed range for setpoint ***SP***.
SysOK	The function completed successfully.

Example:

```
SP2TIME : DateTime;
…
SP2TIME := 08:15:00
SetSPTime (2, SP2TIME);
```

SetSPDuration

For time of day (TOD) setpoints, sets the trip duration (DURATION parameter) of setpoint ***SP*** to the value specified by ***DT***.

Syntax:

```
function SetSPDuration (SP : Integer; DT : DateTime) : SysCode;
```

SysCode values returned:

SysInvalidSetpoint	The setpoint specified by ***SP*** does not exist.
SysInvalidRequest	The setpoint specified by ***SP*** has no DURATION parameter.
SysBatchRunning	The value cannot be changed because a batch process is currently running.
SysOutOfRange	The value specified for ***DT*** is not in the allowed range for setpoint ***SP***.
SysOK	The function completed successfully.

Example:

```
SP3DUR : DateTime;
…
SP3DUR := 00:3:15
SetSPDuration (3, SP3DUR);
```

StartBatch

Starts or resumes a batch run.

Syntax:

```
function StartBatch : SysCode;
```

SysCode values returned:

SysPermissionDenied	The BATCHNG configuration parameter is set to OFF.
SysBatchRunning	A batch process is already in progress.
SysOK	The function completed successfully.

StopBatch

Stops a currently running batch.

Syntax:

```
function StopBatch : SysCode;
```

SysCode values returned:

SysPermissionDenied	The BATCHNG configuration parameter is set to OFF.
SysBatchNotRunning	No batch process is running.
SysOK	The function completed successfully.

PauseBatch

Initiates a latched pause of a running batch process.

Syntax:

```
function PauseBatch : SysCode;
```

SysCode values returned:

SysPermissionDenied	The BATCHNG configuration parameter is set to OFF.
SysBatchRunning	A batch process is already in progress.
SysOK	The function completed successfully.

ResetBatch

Terminates a running, stopped, or paused batch process and resets the batch system.

Syntax:

```
function ResetBatch : SysCode;
```

SysCode values returned:

SysPermissionDenied	The BATCHNG configuration parameter is set to OFF.
SysOK	The function completed successfully.

3.5 Serial I/O

Print

Requests a print operation using the print format specified by ***F***. Output is sent to the port specified in the print format configuration. The print format definition determines which serial port the data is sent to.

Syntax:

```
function Print (F : PrintFormat) : SysCode;
```

PrintFormat values sent:

GrossFmt	Gross format
NetFmt	Net format
TrWInFmt	Truck weigh-in format
TrRegFmt	Truck register format (truck IDs and tare weights)
TrWOutFmt	Truck weigh-out format
SPFmt	Setpoint format
AccumFmt	Accumulator format

SysCode values returned:

SysInvalidRequest	The print format specified by ***F*** does not exist.
SysQFull	The request could not be processed because the print queue is full.
SysOK	The function completed successfully.

Example:

```
Fmtout : PrintFormat;
…
Fmtout := NetFmt
Print (Fmtout);
```

StartStreaming

Starts data streaming for the port number specified by ***P***.

Syntax:

```
function StartStreaming (P : Integer) : SysCode;
```

SysCode values returned:

SysInvalidPort	The port number specified for ***P*** is not valid.
SysInvalidRequest	The port specified for ***P*** is not configured for streaming.
SysOK	The function completed successfully.

Example:

```
StartStreaming (1);
```

StopStreaming

Stops data streaming for the port number specified by ***P***.

Syntax:

```
function StopStreaming (P : Integer) : SysCode;
```

SysCode values returned:

SysInvalidPort	The port number specified for ***P*** is not valid.
SysInvalidRequest	The port specified for ***P*** is not configured for streaming.
SysOK	The function completed successfully.

Example:

```
StopStreaming (1);
```

Write

Writes the text specified in the ***<arg-list>*** to the port specified by ***P***. A subsequent ***Write*** or ***Writeln*** operation will begin where this ***Write*** operation ends; a carriage return is not included at the end of the data sent to the port.

Syntax:

```
function Write (P : Integer; <arg-list>);
```

Example:

```
Write (Port1, "This is a test.");
```

WriteIn

Writes the text specified in the ***<arg-list>*** to the port specified by ***P***, followed by a carriage return. A subsequent ***Write*** or ***Writeln*** operation will begin on the next line.

Syntax:

```
function Write (P : Integer; <arg-list>);
```

Example:

```
Writeln (Port1, "This is another test.");
```

3.6 Digital I/O Control

GetDigin

Sets ***V*** to the value of the digital input assigned to slot ***S***, bit ***D***.

Syntax:

```
function GetDigin (S : Integer; D : Integer; VAR V : Integer) : SysCode;
```

SysCode values returned:

SysInvalidRequest	The slot and bit assignment specified is not a valid digital input.
SysOK	The function completed successfully.

Example:

```
DIGINS0B3 : Integer;
…
GetDigin (0, 3, DIGINS0B3);
Writeln (Port1, "Digin S0B3 status is", DIGINS0B3);
```

GetDigout

Sets ***V*** to the value of the digital output assigned to slot ***S***, bit ***D***.

Syntax:

```
function GetDigout (S : Integer; D : Integer; VAR V : Integer) : SysCode;
```

SysCode values returned:

SysInvalidRequest	The slot and bit assignment specified is not a valid digital output.
SysOK	The function completed successfully.

Example:

```
DIGOUTS0B2 : Integer;
…
GetDigin (0, 2, DIGOUTS0B2);
Writeln (Port1, "Digout S0B2 status is", DIGOUTS0B2);
```

SetDigout

Sets value of the digital output assigned to slot ***S***, bit ***D,*** to the value specified by ***V***.

Syntax:

```
function SetDigout (S : Integer;  D : Integer; V : Integer) : SysCode;
```

SysCode values returned:

SysInvalidRequest	The slot and bit assignment specified is not a valid digital output.
SysOutOfRange	The value ***V*** must be 0 (inactive) or 1 (active).
SysOK	The function completed successfully.

Example:

```
DIGOUTS0B2 : Integer;
…
DIGOUTS0B2 := 0;
SetDigout (0, 2, DIGOUTS0B2);
```

3.7 Display Operations

Display Status

Displays the string ***msg*** in the front panel status message area.

Syntax:

```
procedure DisplayStatus (msg : String);
```

Example:

PromptUser

Opens the alpha entry box and places the string ***msg*** in the user prompt area.

Syntax:

```
function PromptUser (msg : String) : SysCode;
```

SysCode values returned:

SysRequestFailed	The prompt could not be opened.
SysOK	The function completed successfully.

Example:

ClosePrompt

Closes a prompt opened by the PromptUser function.

Syntax:

```
procedure ClosePrompt;
```

Example:

GetEntry

Retrieves the user entry from a programmed prompt.

Syntax:

```
function GetEntry : String;
```

Example:

GetDate

Extracts date information from ***DT*** and places the data in variables ***Year*** , ***Month*** , and ***Day*** .

Syntax:

```
procedure GetDate (DT : DateTime; VAR Year : Integer; VAR Month : Integer; VAR Day : Integer);
```

Example:

GetTime

Extracts time information from ***DT*** and places the data in variables ***Hour*** , ***Minute*** , and ***Second*** .

Syntax:

```
procedure GetDate (DT : DateTime; VAR Hour : Integer; VAR Minute : Integer; VAR
Second : Integer);
```

Example:

SetDate

Sets the date in ***DT*** to the values specified by ***Year*** , ***Month*** , and ***Day*** .

Syntax:

```
function SetDate (VAR DT : DateTime; VAR Year : Integer; VAR Month : Integer;
VAR Day : Integer) : SysCode;
```

SysCode values returned:

SysOK	The function completed successfully.

Example:

SetTime

Sets the time in ***DT*** to the values specified by ***Hour*** , ***Minute*** , and ***Second*** .

Syntax:

```
function SetDate (VAR DT : DateTime; VAR Hour : Integer; VAR Minute : Integer;
VAR Second : Integer) : SysCode;
```

SysCode values returned:

SysOK	The function completed successfully.

Example:

3.8 Display Programming

SetBargraphLevel

Sets the displayed level of bargraph widget ***W*** to the percentage specified by ***Level*** .

Syntax:

```
function SetBargraphLevel (W : Integer; Level : Integer) : SysCode;
```

SysCode values returned:

SysInvalidWidget	The bargraph widget specified by ***W*** does not exist.
SysOK	The function completed successfully.

Example:

SetLabelText

Sets the text of label widget ***W*** to ***S***.

Syntax:

```
function SetLabelText (W : Integer; S : String) : SysCode;
```

SysCode values returned:

SysInvalidWidget	The label widget specified by ***W*** does not exist.
SysOK	The function completed successfully.

Example:

SetNumericValue

Sets the value of numeric widget ***W*** to ***V***.

Syntax:

```
function SetNumericValue (W : Integer; V : Real) : SysCode;
```

SysCode values returned:

SysInvalidWidget	The numeric widget specified by ***W*** does not exist.
SysOK	The function completed successfully.

Example:

SetSymbolState

Sets the state of symbol widget ***W*** to ***V***.

Syntax:

```
function SetSymbolState (W : Integer; S : Integer) : SysCode;
```

SysCode values returned:

SysInvalidWidget	The symbol widget specified by ***W*** does not exist.
SysOK	The function completed successfully.

Example:

SetWidgetVisibility

Sets the visibility state of widget ***W*** to ***V***.

Syntax:

```
function SetSymbolState (W : Integer; V : OnOffType) : SysCode;
```

SysCode values returned:

SysInvalidWidget	The widget specified by ***W*** does not exist.
SysOK	The function completed successfully.

Example:

3.9 Database Operations

<DB>.Add

Adds a record to the referenced database.

Syntax:

```
function <DB>.Add : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysDatabaseFull	There is no space in the specified database for this record.
SysOK	The function completed successfully.

Example:

<DB>.Clear

Clears all records from the referenced database.

Syntax:

```
function <DB>.Clear : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysOK	The function completed successfully.

Example:

<DB>.Delete

Deletes the current record from the referenced database.

Syntax:

```
function <DB>.Delete : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysOK	The function completed successfully.

Example:

<DB>.FindFirst

Finds the first record in the referenced database that matches the contents of ? column ***I***.

Syntax:

```
function <DB>.FindFirst (I : Integer) : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysNoSuchColumn	The column specified by ***I*** does not exist.
SysOK	The function completed successfully.

Example:

<DB>.FindLast

Finds the last record in the referenced database that matches the contents of ? column ***I***.

Syntax:

```
function <DB>.FindLast (I : Integer) : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysNoSuchColumn	The column specified by ***I*** does not exist.
SysOK	The function completed successfully.

Example:

<DB>.FindNext

Finds the next record in the referenced database that matches the criteria of a previous FindFirst or FindLast operation.

Syntax:

```
function <DB>.FindNext : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysOK	The function completed successfully.

Example:

<DB>.FindPrev

Finds the previous record in the referenced database that matches the criteria of a previous FindFirst or FindLast operation.

Syntax:

```
function <DB>.FindLast : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysOK	The function completed successfully.

Example:

<DB>.GetFirst

Retrieves the first logical record from the referenced database.

Syntax:

```
function <DB>.GetFirst : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysOK	The function completed successfully.

Example:

<DB>.GetLast

Retrieves the last logical record from the referenced database.

Syntax:

```
function <DB>.GetLast : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysOK	The function completed successfully.

Example:

<DB>.GetNext

Retrieves the next logical record from the referenced database.

Syntax:

```
function <DB>.GetNext : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysOK	The function completed successfully.

Example:

<DB>.GetPrev

Retrieves the previous logical record from the referenced database.

Syntax:

```
function <DB>.GetPrev : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysOK	The function completed successfully.

Example:

<DB>.Sort

Sorts database ***<DB>*** into ascending order based on the contents of column ***I***.

Syntax:

```
function <DB>.Sort (I : Integer) : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysOK	The function completed successfully.

Example:

<DB>.Update

Updates the current record in the referenced database with the contents of ?.

Syntax:

```
function <DB>.Update : SysCode;
```

SysCode values returned:

SysNoSuchDatabase	The referenced database cannot be found.
SysNoSuchRecord	The requested record is not contained in the database.
SysOK	The function completed successfully.

Example:

3.10 Timer Controls

ResetTimer

Resets the value of timer ***T*** by stopping the timer, setting the timer mode to ***TimerOneShot*** , and setting the timer time-out to 1.

Syntax:

```
function ResetTimer (T : Integer) : Syscode;
```

SysCode values returned:

SysInvalidTimer	The timer specified by ***T*** a not valid timer.
SysOK	The function completed successfully.

SetTimer

Sets the time-out value of timer ***T***. Timer values are specified in 0.01-second intervals (100 = 1 second).

Syntax:

```
function SetTimer (T : Integer ; V : Integer) : Syscode;
```

SysCode values returned:

SysInvalidTimer	The timer specified by ***T*** a not valid timer.
SysOK	The function completed successfully.

SetTimerMode

Sets the mode value, ***M***, of timer ***T***.

Syntax:

```
function SetTimer (T : Integer ; M : TimerMode) : Syscode;
```

TimerMode values sent:

TimerOneShot	Timer mode is set to one-shot.
TimerContinuous	Timer mode is set to continuous.

SysCode values returned:

SysInvalidTimer	The timer specified by ***T*** a not valid timer.
SysInvalidState	The timer mode specified by ***M*** a not valid timer mode.
SysOK	The function completed successfully.

StartTimer

Starts timer ***T***.

Syntax:

```
function StartTimer (T : Integer) : Syscode;
```

SysCode values returned:

SysInvalidTimer	The timer specified by ***T*** a not valid timer.
SysOK	The function completed successfully.

StopTimer

Stops timer ***T***.

Syntax:

```
function StopTimer (T : Integer) : Syscode;
```

SysCode values returned:

SysInvalidTimer	The timer specified by ***T*** a not valid timer.
SysOK	The function completed successfully.

3.11 Mathematical Operations

Abs

Returns the absolute value of ***x***.

Syntax:

```
function Abs (x : Real) : Real;
```

ATan

Returns a value between $-\pi/2$ and $\pi/2$, representing the arctangent of ***x*** in radians.

Syntax:

```
function Atan (x : Real) : Real;
```

Ceil

Returns the smallest integer greater than or equal to ***x***.

Syntax:

```
function Ceil (x : Real) : Real;
```

Cos

Returns the cosine of ***x***. ***x*** must be specified in radians.

Syntax:

```
function Cos (x : Real) : Real;
```

Exp

Returns the value of e^x.

Syntax:

```
function Exp (x : Real) : Real;
```

Log

Returns the value of $\log_e(x)$.

Syntax:

```
function Log (x : Real) : Real;
```

Log10
Returns the value of $\log_{10}(x)$.

Syntax:
```
function Log10 (x : Real) : Real;
```

Sign
Returns the sign of the numeric operand. If $x < 0$, the function returns a value of –1; otherwise, the value returned is 1.

Syntax:
```
function Sign (x : Real) : Integer;
```

Sin
Returns the sine of x. x must be specified in radians.

Syntax:
```
function Sin (x : Real) : Real;
```

Sqrt
Returns the square root of x.

Syntax:
```
function Sqrt (x : Real) : Real;
```

Tan
Returns the tangent of x. x must be specified in radians.

Syntax:
```
function Tan (x : Real) : Real;
```

3.12 String Operations

Asc
Returns the ASCII value of the first character of string S. If S is an empty string, the value returned is 0.

Syntax:
```
function Asc (S : String) : Integer;
```

Chr$
Returns a one-character string containing the ASCII character represented by I.

Syntax:
```
function Chr$ (I : Integer) : String;
```

Hex$
Returns an eight-character hexadecimal string equivalent to I.

Syntax:
```
function Hex$ (I : Integer) : String;
```

LCase$
Returns the string S with all upper-case letters converted to lower case.

Syntax:
```
function LCase$ (S : String) : String;
```

Left$
Returns a string containing the leftmost I characters of string S. If I is greater than the length of S, the function returns a copy of S.

Syntax:
```
function Left$ (S : String; I : Integer) : String;
```

Len
Returns the length (number of characters) of string S.

Syntax:
```
function Len (S : String) : Integer;
```

Mid$

Returns a number of characters (specified by ***length***) from string ***s***, beginning with the character specified by ***start*** . If ***start*** is greater than the string length, the result is an empty string. If ***start + length*** is greater than the length of ***S***, the returned value contains the characters from ***start*** through the end of ***S***.

Syntax:

```
function Mid$ (S : String; start : Integer; length : Integer) : String;
```

Oct$

Returns an 11-character octal string equivalent to ***I***.

Syntax:

```
function Oct$ (I : Integer) : String;
```

Right$

Returns a string containing the rightmost ***I*** characters of string ***S***. If ***I*** is greater than the length of ***S***, the function returns a copy of ***S***.

Syntax:

```
function Right$ (S : String; I : Integer) : String;
```

Space$

Returns a string containing ***N*** spaces.

Syntax:

```
function Space$ (N : Integer) : String;
```

UCase$

Returns the string ***S*** with all lower-case letters converted to upper case.

Syntax:

```
function UCase$ (S : String) : String;
```

3.13 Data Conversion

IntegerToString

Returns a string representation of the integer ***I*** with a minimum length of ***W***. If ***W*** is less than zero, zero is used as the minimum length. If ***W*** is greater than 100, 100 is used as the minimum length.

Syntax:

```
function IntegerToString (I : Integer; W : Integer) : String;
```

RealToString

Returns a string representation of the real number ***R*** with a minimum length of ***W***, with ***P*** digits to the right of the decimal point. If ***W*** is less than zero, zero is used as the minimum length; if ***W*** is greater than 100, 100 is used as the minimum length. If ***P*** is less than zero, zero is used as the precision; if ***P*** is greater than 20, 20 is used.

Syntax:

```
function RealToString (R : Real; W : Integer; P: Integer) : String;
```

StringToInteger

Returns the integer equivalent of the numeric string ***S***. If ***S*** is not a valid string, the function returns the value 0.

Syntax:

```
function StringToInteger (S : String) : Integer;
```

StringToReal

Returns the real number equivalent of the numeric string ***S***. If ***S*** is not a valid string, the function returns the value 0.0.

Syntax:

```
function StringToReal (S : String) : Real;
```

4.0 Programming Examples

4.1 Handler Examples

The following simple programs show examples of event handlers used to initiate other indicator functions. See the list of event handlers in Section 5.1 on page 45.

Example 1: Batch Paused by Digital Input

In the following program, the event handler, `DiginS0B3Activate` is used to pause a running batch whenever the digital input assigned to slot 0, bit 3, goes low. The `StartBatch` function or other input would be required to restart the batch.

```
program DigPause

  handler DiginS0B3Activate;

  begin
    PauseBatch;
  end;
end DigPause;
```

Example 2: Batching Mode Set by Digital Input

In the following program, the event handler `DiginS0B3Activate` is used to set the batching mode to AUTO whenever the digital input assigned to slot 0, bit 3, goes low (active).

The `DiginS0B3Deactivate` handler is used to set the batching state to MANUAL whenever the digital input goes high (inactive). The batching mode change takes place immediately, even if a batch sequence is in progress.

```
program AutoMan

  handler DiginS0B3Activate;

  begin
    SetBatchingMode(AUTO);
  end;

  handler DiginS0B3Deactivate;

  begin
    SetBatchingMode(MANUAL);
  end;

end AutoMan;
```

Example 3: Serial Notification of Setpoint Trip

The following program uses the event handler `SP1Trip` to send a notification to Port 1 that the setpoint has tripped.

```
program SPDebug

  handler SP1Trip;

  begin
    Writeln (1, "SP 1 TRIPPED");
  end;
end SPDebug;
```

Example 4: Setpoints Toggled by Softkeys

In the following program, the event handlers `User2KeyPressed` and `User4KeyPressed` are used to turn groups of setpoints on and off. When the second softkey is pressed, setpoints 1 and 3 are turned off, setpoints 2 and 4 are turned on. When the fourth softkey is pressed, setpoints 1 and 3 are turned on, 2 and 4 are turned off.

```
program SPTURN

  handler User2KeyPressed;

  begin
    DisableSP(1);
    DisableSP(3);
    EnableSP(2);
    EnableSP(4);

  end;

  handler User4KeyPressed;

  begin
    DisableSP(2);
    DisableSP(4);
    EnableSP(1);
    EnableSP(3);
  end;

end SPTURN;
```

4.2 Hello

```
program Hello;

  --
  -- Define constants for use in the program
  --

  Port1 : constant integer := 1;
  Greeting : constant string := "HELLO 920i USER!!!";

--
--  User1KeyPressed handler to display message on
--  920i Status line and write the message out RS232
--  Port 1.
--

handler User1KeyPressed;

  begin
    DisplayStatus(Greeting);
    WriteLn(Port1, Greeting);
  end;

--
--  Clear the status line with the User2KeyPressed handler
--

handler User2KeyPressed;

  begin
    DisplayStatus("");
  end;

end Hello;
```

4.3 Looptest

```
handler User1KeyPressed;

  LoopCounter : integer;

  begin
    for LoopCounter := 1 to 10
    loop
      WriteLn(Port1, IntegerToString(LoopCounter, 0));
    end loop;
  end;
------------------------------------------------------
--
-- Second loop test counts from 10 to 1 with a 'while' loop
-- and writes the values out a RS232 port.
--
------------------------------------------------------
handler User2KeyPressed;

  LoopCounter : integer;

  begin
    LoopCounter := 10;
    while LoopCounter > 1
    loop
      WriteLn(Port1, LoopCounter);
      LoopCounter := LoopCounter - 1;
    end loop;
  end;

------------------------------------------------------
--
--  Third loop test counts from 10 to 1 with no 'for' or 'while'
--  condition. Notice the 'exit' statement is required to stop
--  the loop.  Otherwise the system will stay in the loop, NOT GOOD.
--
------------------------------------------------------
handler User3KeyPressed;

  LoopCounter : integer;

  begin
    LoopCounter := 10;
    loop
      WriteLn(Port1, LoopCounter);
      LoopCounter := LoopCounter - 1;
      if LoopCounter < 1 then
        exit;
      end if;
    end loop;
  end;

end LoopTest;
```

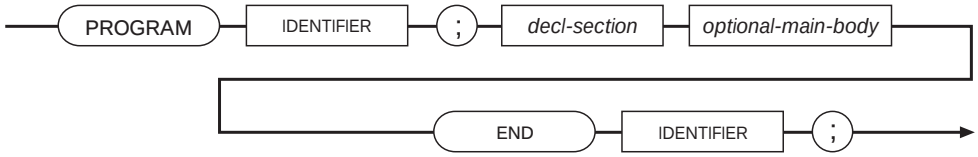
5.0 Appendix

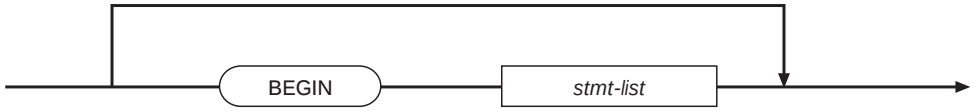
5.1 Event Handlers

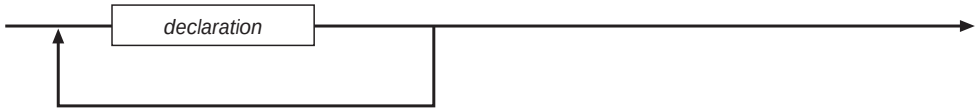
Handler	Description
KeyPressed	Runs when any front panel key is pressed
ZeroKeyPressed	Runs when the ZERO key is pressed
GrossNetKeyPressed	Runs when the GROSS/NET key is pressed
TareKeyPressed	Runs when the TARE key is pressed
UnitsKeyPressed	Runs when the UNITS key is pressed
PrintKeyPressed	Runs when the PRINT KEY is pressed
MajorKeyPressed	Runs when any of the five preceding major keys is pressed
Soft*x*KeyPressed	Runs when softkey *x* is pressed, where x=the softkey number, 1–5, left to right
SoftKeyPressed	Runs when any softkey is pressed
N*x*KeyPressed	Runs when a numeric key is pressed, where *x*=the key number 0–9
DotKeyPressed	Runs when the decimal point key on the numeric keypad is pressed
ClearKeyPressed	Runs when the CLR key on the numeric keypad is pressed
NumericKeyPressed	Runs when any key on the numeric keypad (including CLR or decimal point) is pressed
NavUpKeyPressed	Runs when the UP navigation key is pressed
NavDownKeyPressed	Runs when the DOWN navigation key is pressed
NavLeftKeyPressed	Runs when the LEFT navigation key is pressed
NavRightKeyPressed	Runs when the RIGHT navigation key is pressed
EnterKeyPressed	Runs when the ENTER key on the front panel is pressed
NavKeyPressed	Runs when any of the navigation cluster keys (including ENTER) is pressed
SP*x*Trip	Runs when setpoint *x* is tripped, where *x* is the setpoint number, 1…100)
DiginS*x*B*y*Activate	Runs when the digital input assigned to slot *x*, bit *y* is activated. Valid bit assignments for slot 0 are 1–4; valid bit assignments for slots 1 through 14 are 1–24.
DiginS*x*B*y*Deactivate	Runs when the digital input assigned to slot *x*, bit *y* is activated. Valid bit assignments for slot 0 are 1–4; valid bit assignments for slots 1 through 14 are 1–24.
Port*x*CharReceived	Runs when a character is received on port *x*, where *x* is the port number, 1…32
Timer*x*Trip	Runs when timer *x* is tripped, where *x* is the timer number, 1…32
ProgramStartup	Runs when the indicator is powered-up or when exiting setup mode

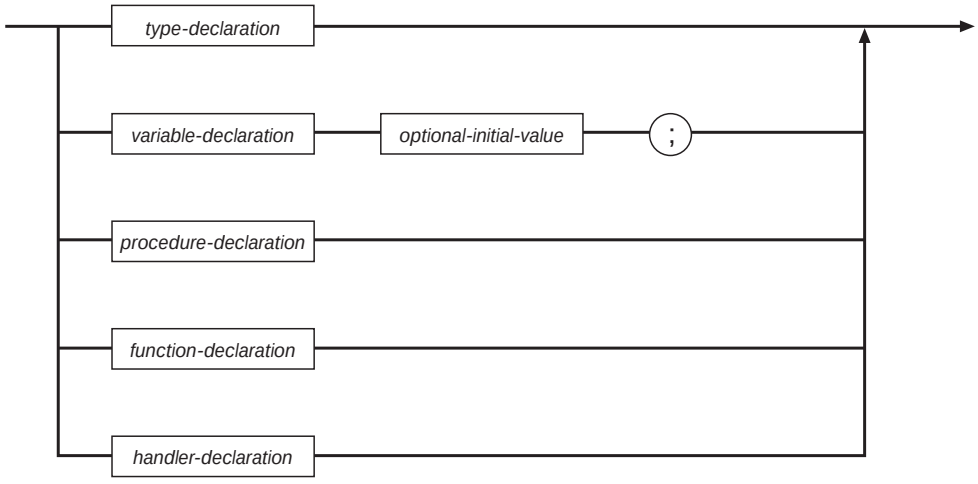
Table 5-1. 920i Event Handlers

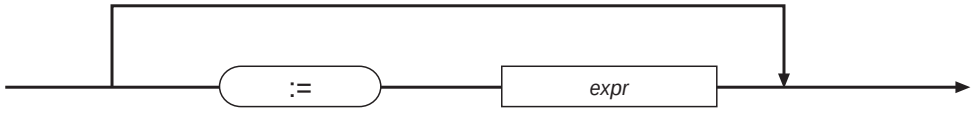
5.2 Syntax Diagrams

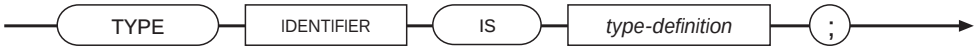


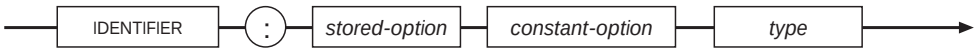


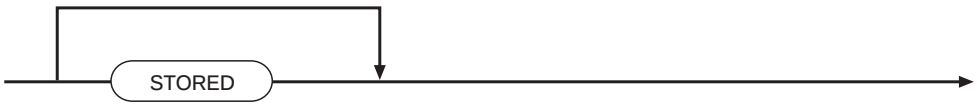




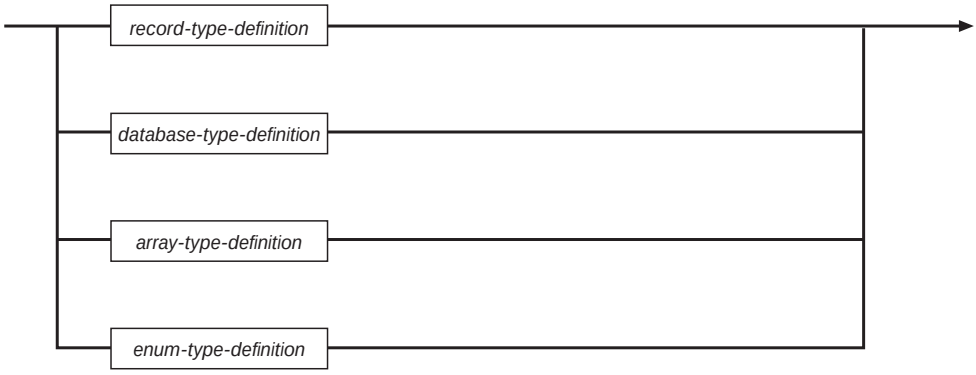



TYPE
IDENTIFIER
IS
type-definition
;

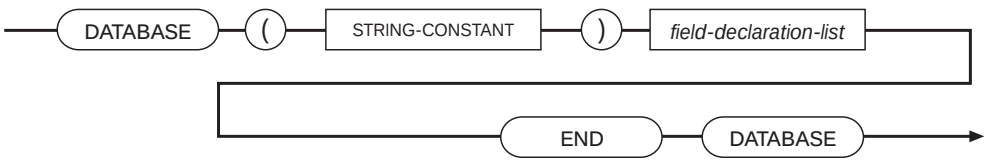

IDENTIFIER
:
stored-option
constant-option
type

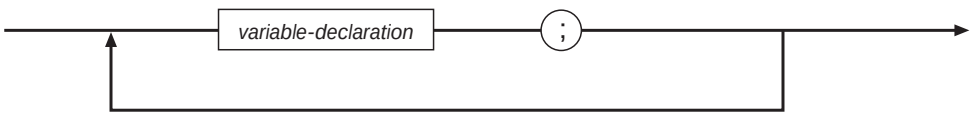

STORED

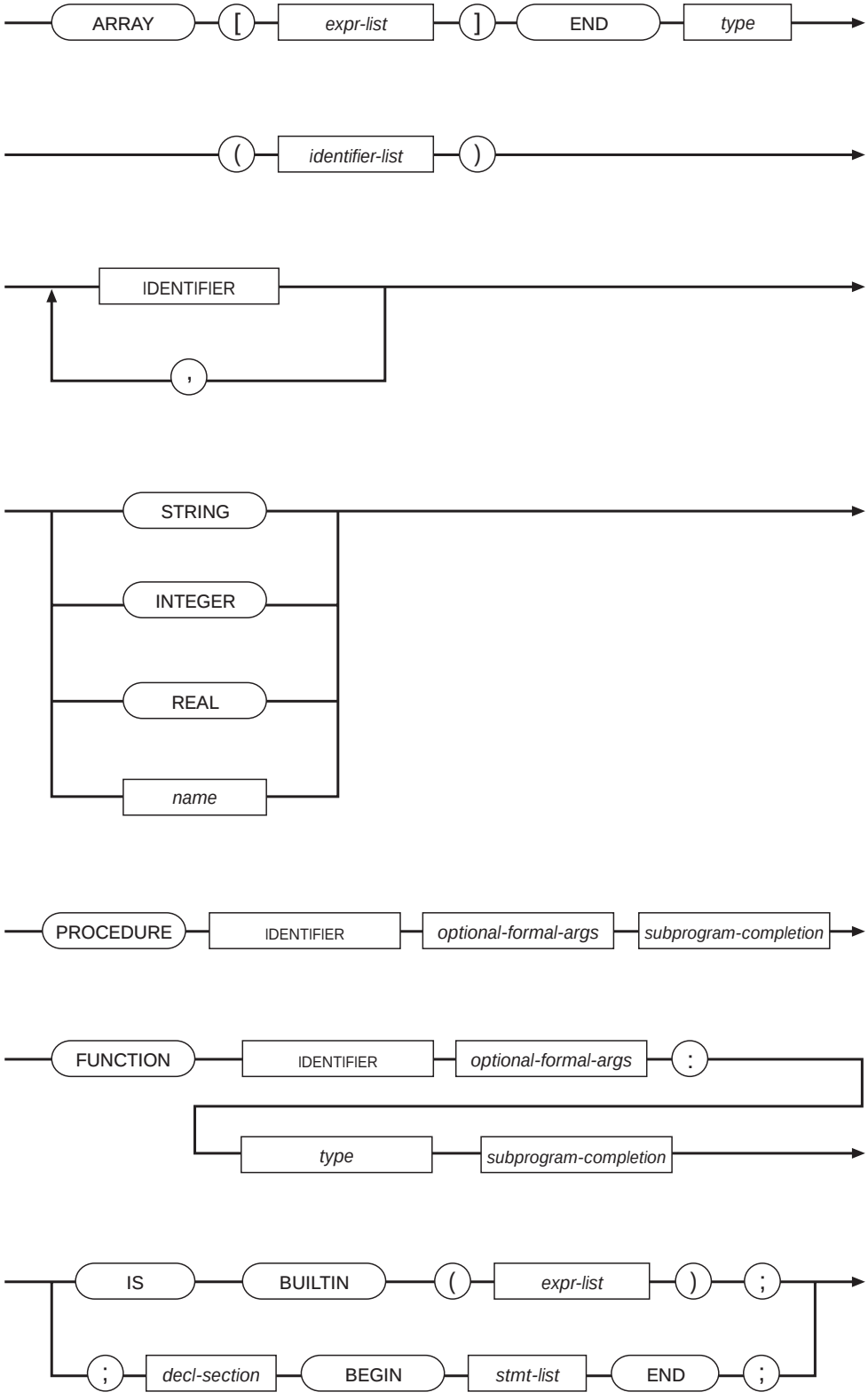

CONSTANT

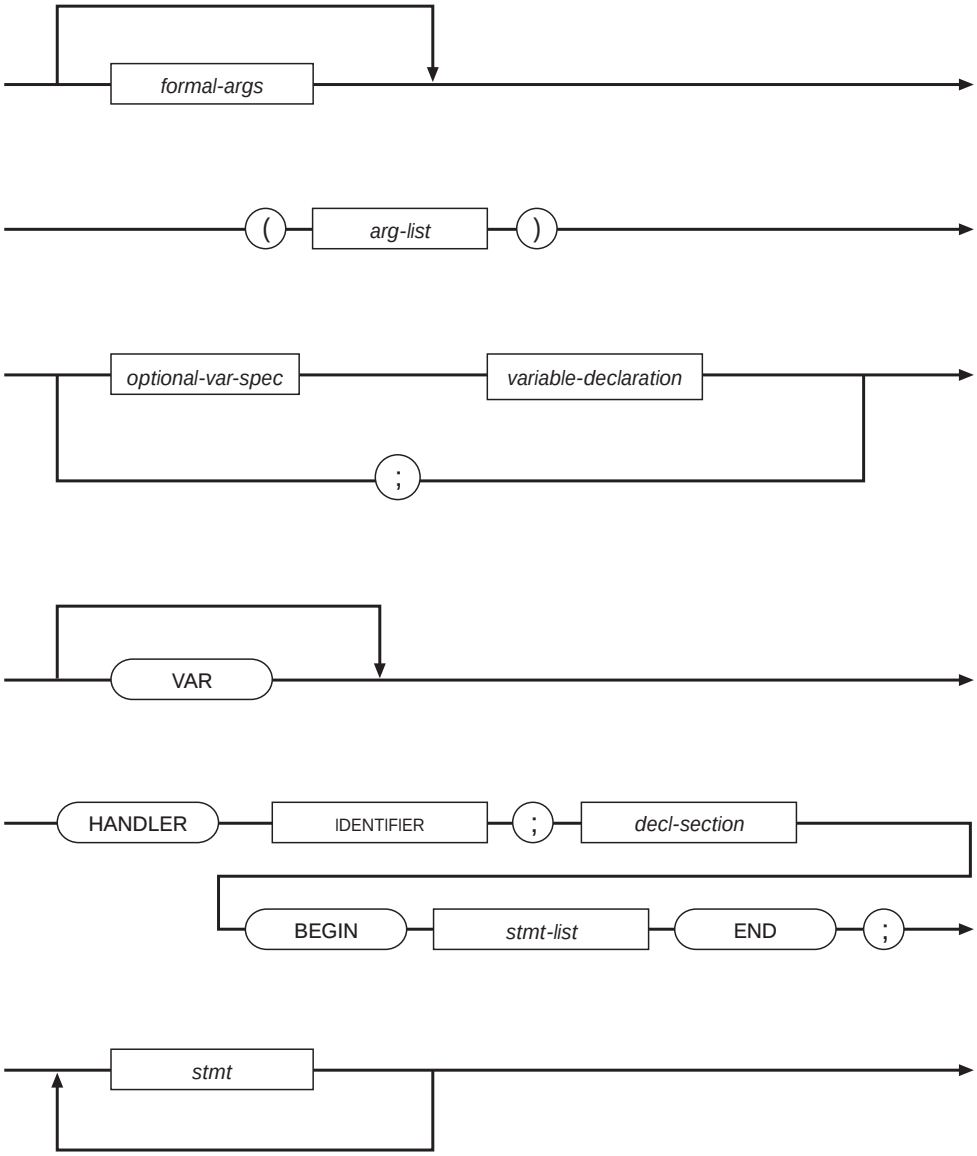

record-type-definition
database-type-definition
array-type-definition
enum-type-definition

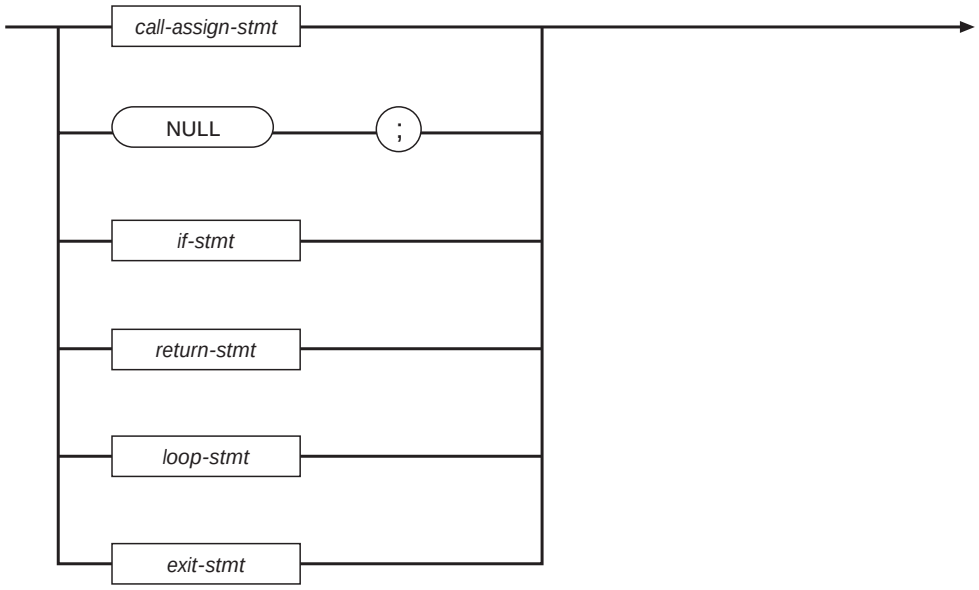

RECORD
field-declaration-list
END
RECORD

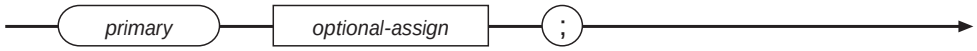

DATABASE
(
STRING-CONSTANT
)
field-declaration-list
END
DATABASE

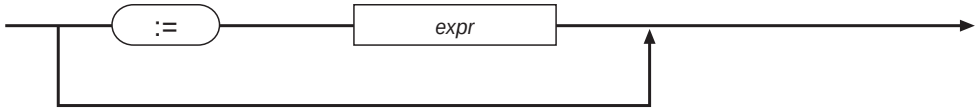

variable-declaration
;

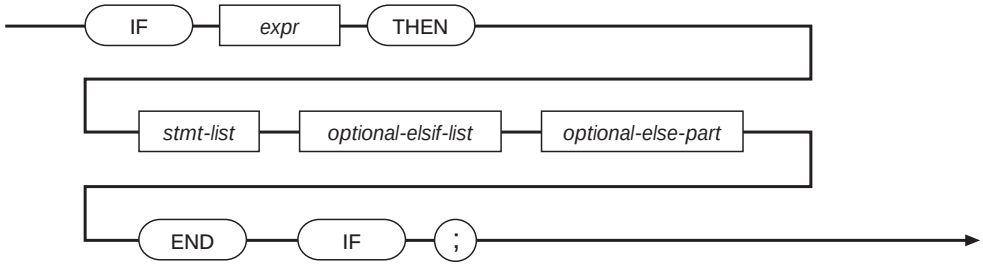

ARRAY
[
expr-list
]
END
type
(
identifier-list
)
IDENTIFIER
,
STRING
INTEGER
REAL
name
PROCEDURE
IDENTIFIER
optional-formal-args
subprogram-completion
FUNCTION
IDENTIFIER
optional-formal-args
:
type
subprogram-completion
IS
BUILTIN
(
expr-list
)
;
;
decl-section
BEGIN
stmt-list
END
;

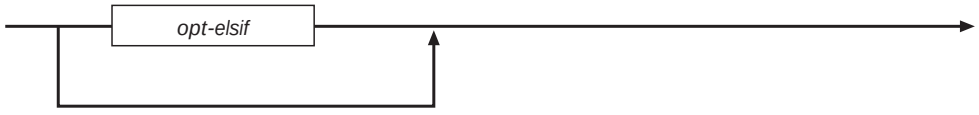

formal-args
(
arg-list
)
optional-var-spec
variable-declaration
;
VAR
HANDLER
IDENTIFIER
;
decl-section
BEGIN
stmt-list
END
;
stmt

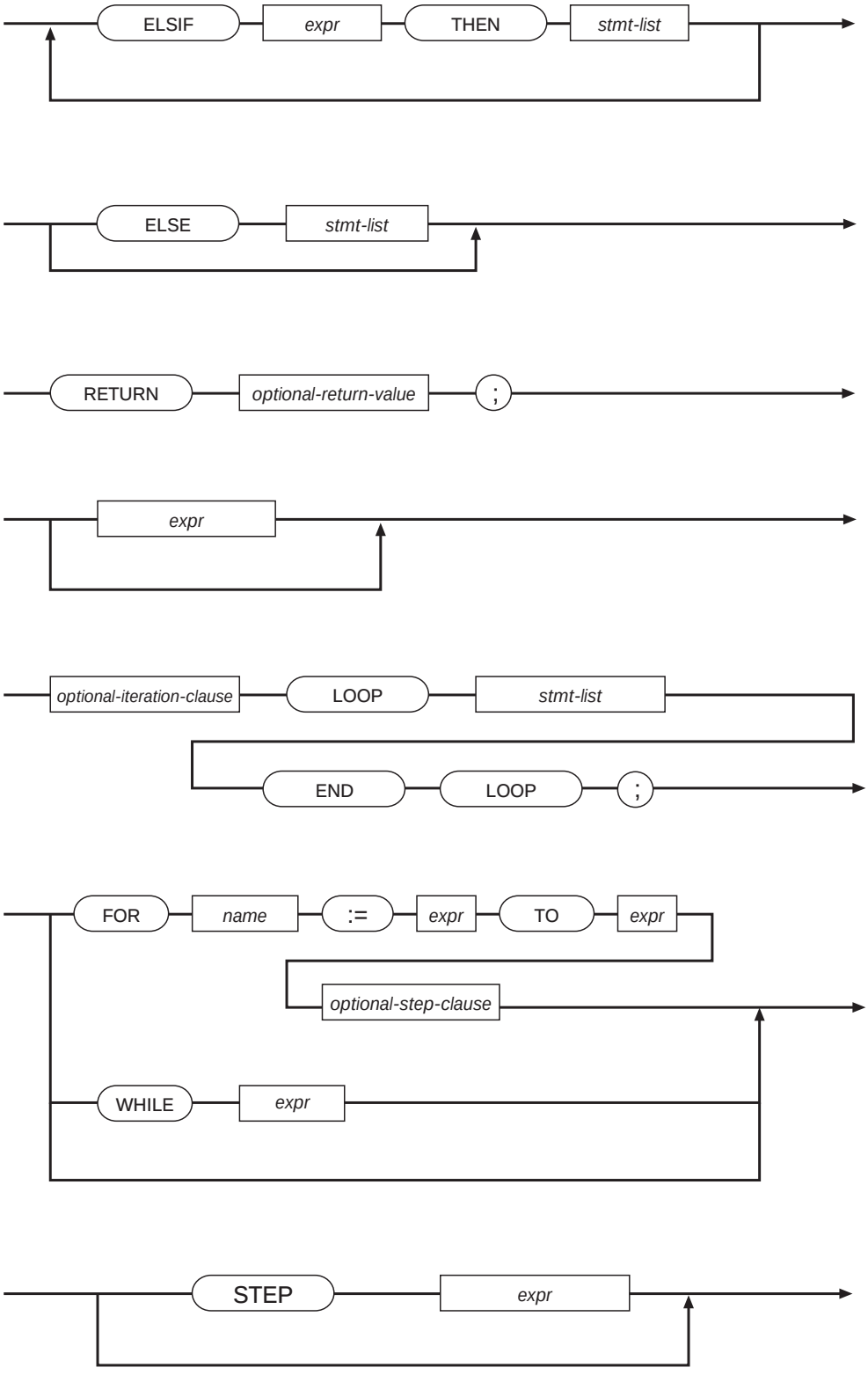

call-assign-stmt
NULL
;
if-stmt
return-stmt
loop-stmt
exit-stmt

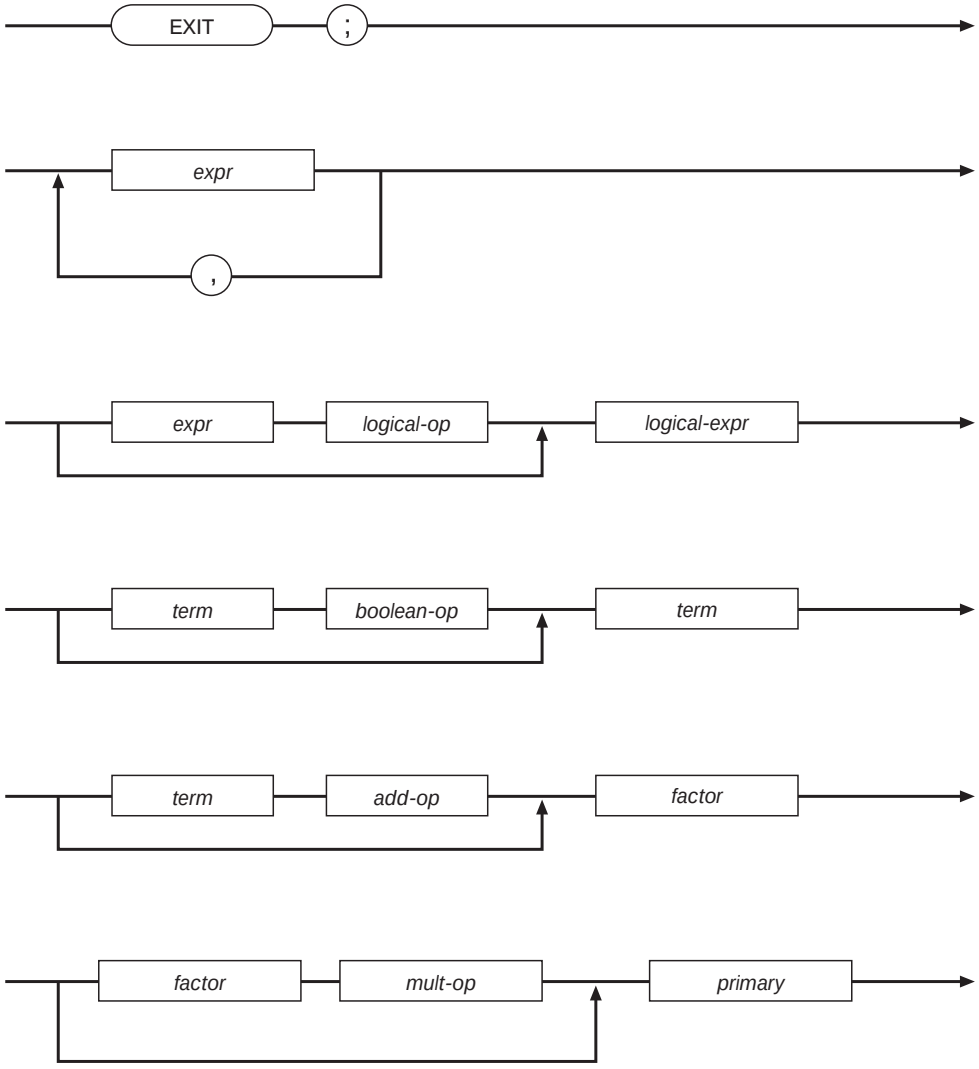

primary
optional-assign
;

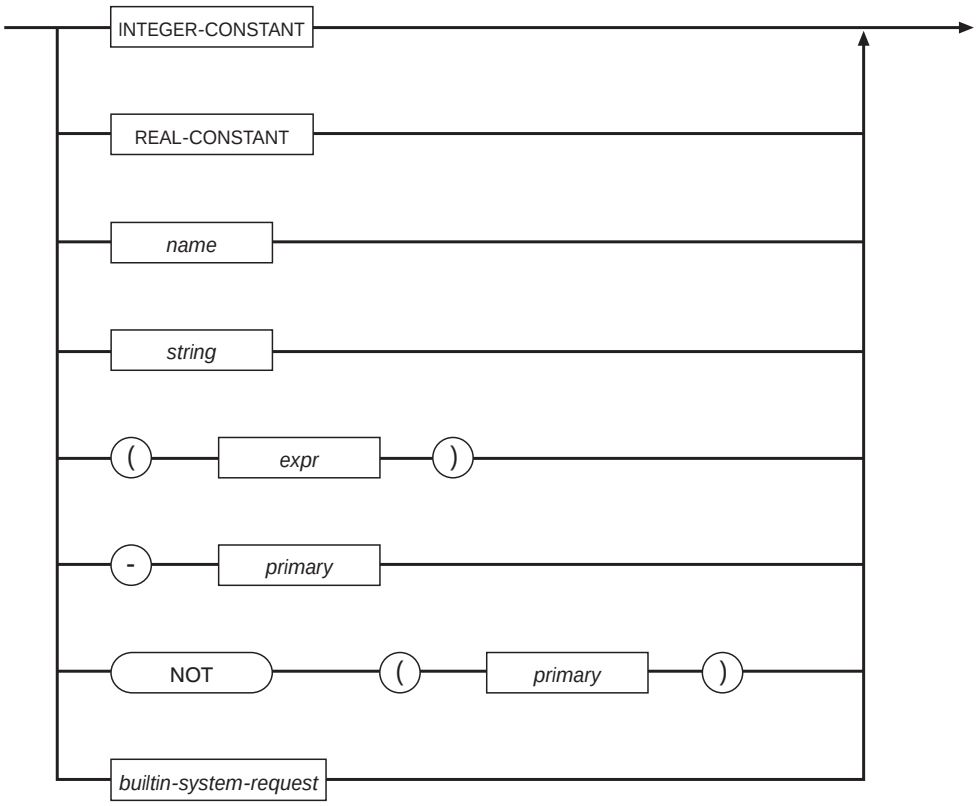

:=
expr

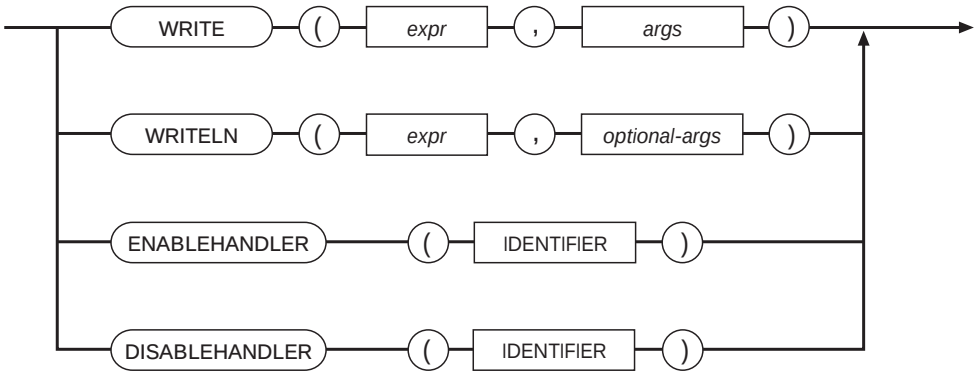

IF
expr
THEN
stmt-list
optional-elsif-list
optional-else-part
END
IF
;

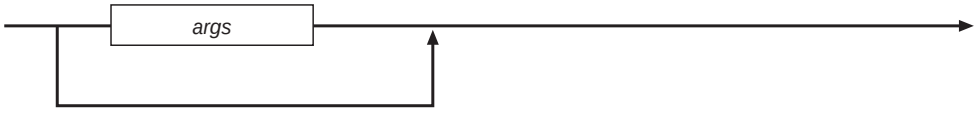

opt-elsif

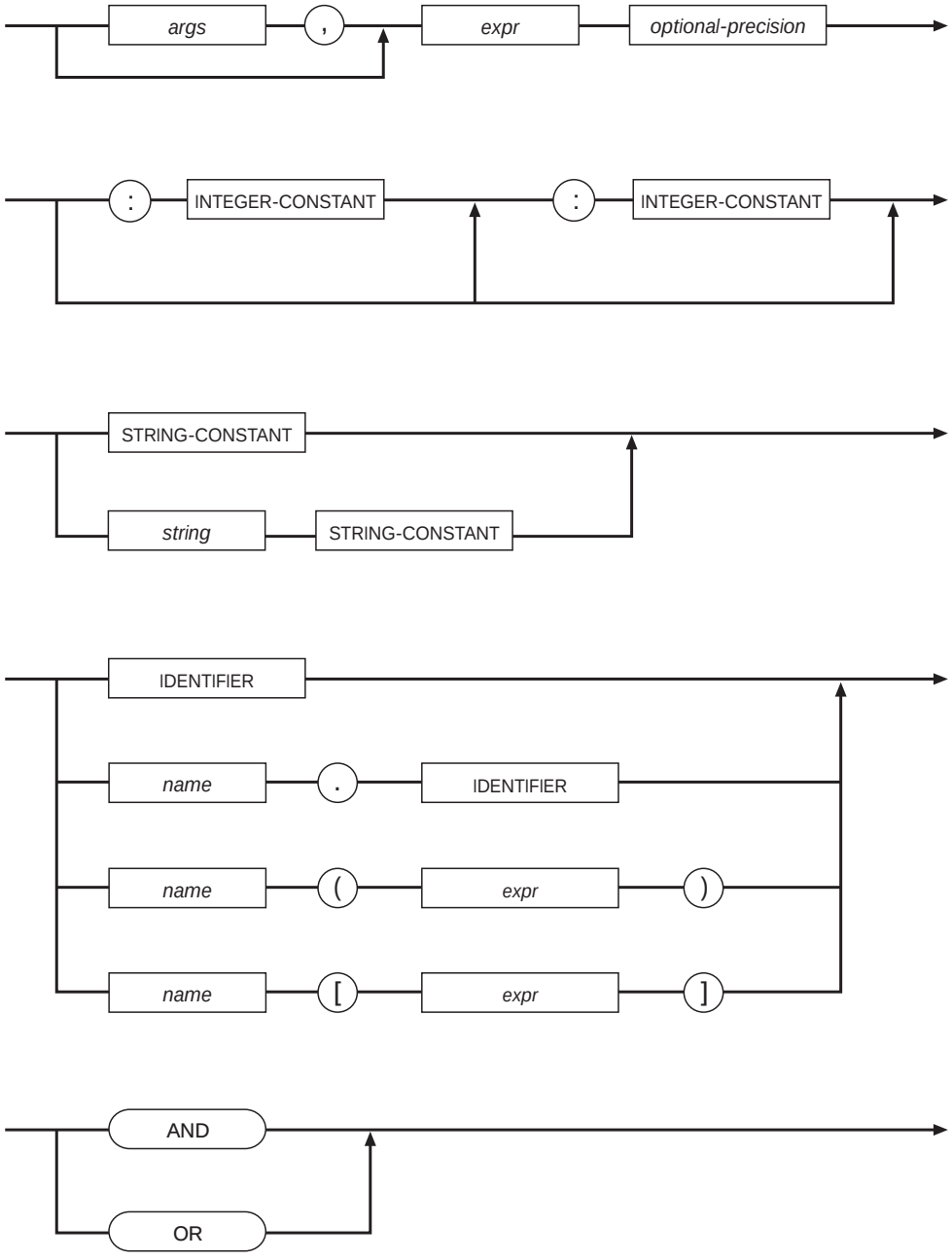

ELSIF
expr
THEN
stmt-list
ELSE
stmt-list
RETURN
optional-return-value
;
expr
optional-iteration-clause
LOOP
stmt-list
END
LOOP
;
FOR
name
:=
expr
TO
expr
optional-step-clause
WHILE
expr
STEP
expr

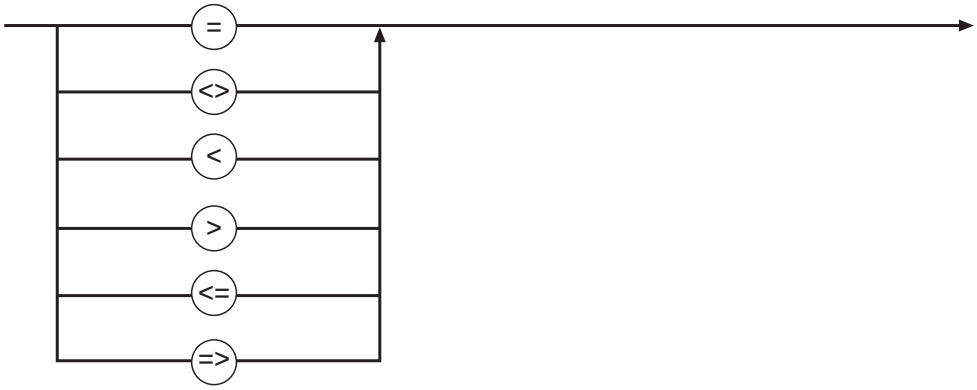

EXIT
;
expr
,
expr
logical-op
logical-expr
term
boolean-op
term
term
add-op
factor
factor
mult-op
primary

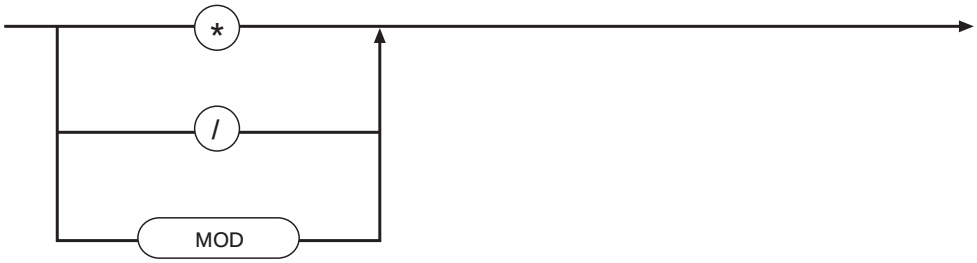

INTEGER-CONSTANT
REAL-CONSTANT
name
string
(
expr
)
-
primary
NOT
(
primary
)
builtin-system-request

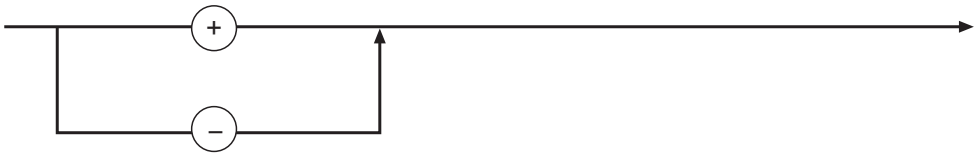

WRITE
(
expr
,
args
)
WRITELN
(
expr
,
optional-args
)
ENABLEHANDLER
(
IDENTIFIER
)
DISABLEHANDLER
(
IDENTIFIER
)


args


args
,
expr
optional-precision
:
INTEGER-CONSTANT
:
INTEGER-CONSTANT
STRING-CONSTANT
string
STRING-CONSTANT
IDENTIFIER
name
.
IDENTIFIER
name
(
expr
)
name
[
expr
]
AND
OR


=
<>
<
>
<=
=>


*
/
MOD


+
–