@

GNU Chill

William Cox, Per Bothner, Wilfried Moser

1 Compiler options

Invoking the compiler:

The GNU CHILL compiler supports several new command line options, and brings a new use to another:

`-lang-chill`
: This option instructs gcc that the following file is a CHILL source file, even though its extension is not the default '.ch'.

`-flocal-loop-counter`
: The CHILL compiler makes a separate reach, or scope, for each DO FOR loop. If `-flocal-loop-counter` is specified, the loop counter of value enumeration and location enumeration is automatically declared inside that reach. This is the default behavior, required by Z.200.

`-fno-local-loop-counter`
: When this option is specified, the above automatic declaration is not performed, and the user must declare all loop counters explicitly.

`-fignore-case`
: When this option is specified, the compiler ignores case. All identifiers are converted to lower case. This enables the usage of C runtime libraries.

`-fno-ignore-case`
: Ignoring the case of identifiers is turned off.

`-fruntime-checking`
: The CHILL compiler normally generates code to check the validity of expressions assigned to variables or expressions passed as parameters to procedures and processes, if those expressions cannot be checked at compile time. This is the default behavior, required by Z.200. This option allows you to re-enable the default behavior after disabling it with the `-fno-runtime-checking` option.

`-fno-runtime-checking`
: The CHILL compiler normally generates code to check the validity of expressions assigned to variables, or expressions passed as parameters to procedures and processes. This option allows you to disable that code generation. This might be done to reduce the size of a program's generated code, or to increase its speed of execution. Compile time range-checking is still performed.

`-fgrant-only`
`-fchill-grant-only`
: This option causes the compiler to stop successfully after creating the grant file specified by the source file (see modular programming in CHILL). No code is generated, and many categories of errors are not reported.

`-fold-string`
: Implement the semantics of Chill 1984 with respect to strings: String indexing yields a slice of length one; CHAR is similar to CHAR(1) (or CHARS(1)); and BOOL is similar to BIT(1) (or BOOLS(1)).

`-fno-old-string`
: Don't implement 1984 Chill string semantics. This is the default.

`-I`*seize_path*
: This directive adds the specified seize path to the compiler's list of paths to search for seize files. When processing a USE_SEIZE_FILE directive, the compiler normally searches for the specified seize file only in the current directory. When one or more seize paths are specified, the compiler also searches in those directories, in the order of their specification on the command line, for the seize file.

`-c`
: This C-related switch, which normally prevents gcc from attempting to link, is *not* yet implemented by the `chill` command, but you can use the `gcc` command with this flag.

2 Implemented and missing parts of the Chill language

The numbers in parentheses are Z.200(1988) section numbers.

- The FORBID keyword in a GRANT statement is currently ignored.
- A CASE action or expression allows only a single expression in a case selector list (5.3.2, 6.4).
- ROW modes are not implemented (3.6.3, 3.13.4).
- Due to the absence of ROW modes, DYNAMIC has no meaning in connection with access and text modes.
- Array and structure layout (PACK, POS, NOPACK, STEP keywords) is ignored (3.12.6).
- Bit-string slices are not implemented.
- The support for synchronization modes and concurrent execution is slightly non-standard.
- Exception handling is implemented, but exceptions are not generated in all of the required situations.
- Dynamic modes are not implemented (though string slices should work).
- Reach-bound initializations are not implemented (4.1.2).

3 GNU-specific enhancements to the Chill language

- Grantfiles. See See Chapter 5 [Separate compilation], page 6.
- Precisions. Multiple integer and real precisions are supported, as well as signed and unsigned variants of the integer modes.
- DESCR built-in. The new built-in function DESCR (<descriptor argument>) returns a pointer to STRUCT(addr PTR, length ULONG) where <descriptor argument> can be anything the compiler can handle but at least a location of any mode (except synchronizing modes) and any character string or powerset value. (A temporary location within the current stack frame may be allocated if an expression is used.)

  CHILL does not permit the writing of procedures with parameters of any type. Yet some interfaces—in particular those to system calls—require the handling of a wide range of modes, e.g. any string mode, any structure mode, or any powerset mode. This could be handled by specifying two parameters (PTR, INT for the length) but this is error-prone (no guarantee the same location is used after in ADDR and LENGTH), and it will not be possible for expressions.

  Caveats: This feature permits the programmer to obtain the address of a literal (if the compiler takes this shortcut—see 1st example below). If hardware features protect constant parts of the program, erronous abuse will be detected.

  Examples: OFFER_HANDLER(descr("dbs"), ->dbs);

  SYNMODE m_els = SET(ela, elb, elc); SYNMODE m_elsel = POWERSET m_els; DCL user_buf STRUCT(a mx, b my, c mz); DCL select POWERSET m_elsel;

  select := m_elsel[LOWER(m_els) : UPPER(m_els)];

  GET_RECORD(relation, recno, descr(user_buf), descr(select));

  PUT_RECORD(relation, recno, descr(user_buf.b), descr(m_elsel[elb]));
- LENGTH built-in on left-hand-side. The LENGTH built-in may be used on the left-hand-side of an assignment, where its argument is a VARYING character string.

4 Value and location conversions

Value and location conversions are highly dependent on the target machine. They are also very loosely specified in the 1988 standard. (The 1992 standard seems an improvement.)

The GNU Chill compiler interprets `mode(exp)` as follows:

- If *exp* is a referable location, and the size of (the mode of) *exp* is the same as the size of *mode*, a location conversion is used. It is implemented exactly as: `(refmode(-> exp))->`, where *refmode* is a synmode for `REF` *mode*.

  The programmer is responsible for making sure that alignment restrictions on machine addresses are not violated.

  If both *mode* and the mode of *exp* are discrete modes, alignment should not be a problem, and we get the same conversion as a standard value conversion.
- If *exp* is a constant, and the size of (the mode of) *exp* is the same as the size of *mode*, then a value conversion is performed. This conversion is done at compile time, and it has not been implemented for all types. Specifically, converting to or from a floating-point type is not implemented.
- If both *mode* and the mode of *exp* are discrete modes, then a value conversion is performed, as described in Z.200.
- If both *mode* and the mode of *exp* are reference modes, then a value conversion is allowed. The same is true is one mode is a reference mode, and the other is an integral mode of the same size.

5 Separate compilation

The GNU CHILL compiler supports modular programming. It allows the user to control the visibility of variables and modes, outside of a MODULE, by the use of GRANT and SEIZE directives. Any location or mode may be made visible to another MODULE by GRANTing it in the MODULE where it is defined, and SEIZEing it in another MODULE which needs to refer to it.

When variables are GRANTed in one or more modules of a CHILL source file, the compiler outputs a grant file, with the original source file name as the base name, and the extension '.grt'. All of the variables and modes defined in the source file are written to the grant file, together with any use_seize_file directives, and the GRANT directives. A grant file is created for every such source file, except if an identical grant file already exists. This prevents unnecessary makefile activity.

The referencing source file must:

1. specify the grant file in a use_seize_file directive, and
2. SEIZE each variable or mode definition that it needs.

An attempt to SEIZE a variable or mode which is not GRANTed in some seize file is an error.

An attempt to refer to a variable which is defined in some seize file, but not explicitly granted, is an error.

An attempt to GRANT a variable or mode which is not defined in the current MODULE is an error.

Note that the GNU CHILL compiler will *not* write out a grant file if:

- there are no GRANT directives in the source file, or
- the entire grant file already exists, and is identical to the file which the compiler has just built. (This latter "feature" may be removed at some point.)

Otherwise, a grant file is an automatic, unsuppressable result of a successful CHILL compilation.

A future release will also support using remote spec modules in a similar (but more Blue Book-conforming) manner.

6 Differences to Z.200/1988

This chapter lists the differences and extensions between GNUCHILL and the CCITT recommendation Z.200 in its 1988 version (reffered to as Z.200/1988).

- 2.2 Vocabulary
  The definition of <*simple name string*> is changed to:

  <*simple name string*> *::=*
  {<*letter*> | _ } { <*letter*> | <*digit* | _ }

- 2.6 Compiler Directives
  Only one directive is allowed between the compiler directive delimiters '<>' and '<>' or the end-of-line, i.e.

```
<> USE_SEIZE_FILE "foo.grt" <>
<> ALL_STATIC_OFF
```

- 3.3 Modes and Classes
  The syntax of <*mode*> is changed to:

  <*mode*> *::=*
  [READ] <*non-composite-mode*>
  | **[READ]** *composite-mode*>

  <*non-composite-mode*> *::=*
  <*discrete mode*>
  | <*real modes*>
  | <*powerset modes*>
  | <*reference mode*>
  | <*procedure mode*>
  | <*instance mode*>
  | <*synchronization mode*>
  | <*timing mode*>

- 3.4 Discrete Modes
  The list of discrete modes is enhanced by the following modes:

```
BYTE        8-bit signed integer
UBYTE       8-bit unsigned integer
UINT        16-bit unsigned integer
LONG        32-bit signed integer
ULONG       32-bit unsigned integer
```

  Please note that INT is implemented as 16-bit signed integer.

- 3.4.6 Range Modes
  The mode BIN(n) is not implemented. Using INT(0 : 2 ** n - 1) instead of BIN(n) makes this mode unneccessary.

- 3.X Real Modes
  Note: This is an extension to Z.200/1988, however, it is defined in Z.200/1992.

  syntax:

  <*real mode*> *::=*
  <*floating point mode*>

  semantics:

```
A real mode specifies a set of numerical values which approximate a
contiguous range of real numbers.
```

- 3.X.1 Floating point modes

  syntax:

  <*floating point mode*> *::=*
      <*floating point mode name*

  predefined names:

  The names *REAL* and *LONG_REAL* are predefined as **floating point mode** names.

  semantics:

  A floating point mode defines a set of numeric approximations to a range of real values, together with their minimum relative accuracy, between implementation defined bounds, over which the usual ordering and arithmetic operations are defined. This set contains only the values which can be represented by the implementation.

  examples:

  REAL
  LONG_REAL

- 3.6 Reference Modes
  Row modes are not implemeted at all.
- 3.7 Procedure Mode
  The syntax for procedure modes is changed to:

  <*procedure mode*> *::=*
        PROC *([*<*parameter list*>*]) [* <*result spec*> *]*
        *[***EXCEPTIONS***(*<*exception list*>*)] [***RECURSIVE***]*
      | <*procedure mode name*>

  <*parameter list*> *::=*
      <*parameter spec*> {, <*parameter spec*> } *

  <*parameter spec*> *::=*
      <*mode*> *[* <*parameter attribute*> *]*

  <*parameter attribute*> *::=*
      IN | **OUT** | **INOUT** | **LOC**

  <*result spec*> *::=*
      RETURNS *(* <*mode*> *[***LOC***])*

  <*exception list*> *::=*
      <*exception name*> {, <*exception name*> } *

- 3.10 Input-Output Modes
  Due to the absence of row modes, DYNAMIC has no meaning in an access or text mode definition.

- 3.12.2 String Modes
  As <*string modes*> were defined differently in Z.200/1984, the syntax of <*string mode*> is changed to:

  <*string mode*> *::=*
  <*string type*> *(* <*string length*> *) [* **VARYING** *]*
  | <*parametrized string mode*>
  | <*string mode name*>

  <*parameterized string mode*> *::=*
  <*origin string mode name*> *(* <*string length*> *)*
  | <*parameterized string mode name*>

  <*origin string mode name*> *::=*
  <*string mode name*>

  string type
  BOOLS
  | **BIT**
  | **CHARS**
  | **CHAR**

  <*string length*> *::=*
  <*integer literal expression*>

  VARYING is not implemented for <*string type*> **BIT** and **BOOL**.
- 3.11.1 Duration Modes
  The predefined mode *DURATION* is implemented as a NEWMODE ULONG and holds the duration value in miliseconds. This gives a maximum duration of

  ```
  MILLISECS (UPPER (ULONG)),
  SECS (4294967),
  MINUTES (71582),
  HOURS (1193), and
  DAYS (49).
  ```
- 3.11.2 Absolute Time Modes
  The predefined mode *TIME* is implemented as a NEWMODE ULONG and holds the absolute time in seconds since Jan. 1st, 1970. This is equivalent to the mode 'time_t' defined on different systems.
- 3.12.4 Structure Modes
  Variant fields are allowed, but the CASE-construct may define only one tag field (one dimensional CASE). OF course, several variant fields may be specified in one STRUCT mode. The tag field will (both at compile- and runtime) not be interpreted in any way, however, it must be interpreted by a debugger. As a consequence, there are no parameterized STRUCT modes.
- 3.12.5 Layout description for array and structure modes
  STEP and POS is not implemeted at all, therefore the syntax of <*element layout* and *field layout* is changed to:

  <*element layout*> *::=*

PACK | **NOPACK**

<*field layout*> *::=*
PACK | **NOPACK**

- 3.13.4 Dynamic parameterised structure modes
  Dynamic parameterised structure modes are not implemented.
- 4.1.2 Location declaration
  The keyword STATIC is allowed, but has no effect at module level, because all locations declared there are assumed to be 'static' by default. Each granted location will become 'public'. A 'static' declaration inside a block, procedure, etc. places the variable in the data section instead of the stack section.
- 4.1.4 Based decleration
  The based declaration was taken from Z.200/1984 and has the following syntax:

  syntax:

  <*based declaration*> *::=*
  <*defining occerrence list*> <*mode*> **BASED**
  (<*free reference location name*> *)*

  semantics:

  A based declaration with <*free reference location name*> specifies as many access names as are defining occerrences in the *defining occurrence list*. Names declared in a base declaration serve as an alternative way accessing a location by dereferencing a reference value. This reference value is contained in the location specified by the *free reference location name*. This dereferencing operation is made each time and only when an access is made via a declared **based** name.

  static properties:

  A defining occurrence in a *based declaration* with *free reference location name* defines a **based** name. The mode attached to a **based** name is the *mode* specified in the *based declaration*. A **based** name is **referable**.
- 4.2.2 Access names
  The syntax of access names is changed to:

  <*access name*> *::=*
  <*location name*>
  | <*loc-identity name*>
  | <*based name*>
  | <*location enumeration name*>
  | <*location do-with name*>

  The semantics, static properties and dynamic conditions remain unchanged except that they are enhanced by *base name*.
- 5.2.4.1 Literals General
  The syntax of <*literal*> is change to:

  <*literal*> *::=*
  <*integer literal*>
  | <*boolean literal*>
  | <*charater literal*>

| *<set literal>*
| *<emptiness literal>*
| *<character string literal>*
| *<bit string literal>*
| *<floating point literal>*

Note: The *<floating point literal>* is an extension to Z.200/1988 and will be described later on.

- 5.2.4.2 Integer literals
  The *<decimal integer literal>* is changed to:

  <decimal integer literal> ::=
  { *D* | *d* } ' { *<digit>* | _ } +
  | *<digit>* { *<digit>* | _ } *

- 5.2.4.4 Character literals
  A character literal, e.g. 'M', may serve as a charater string literal of length 1.
- 5.2.4.7 Character string literals
  The syntax of a character string literal is:

  <character string literal> ::=
  '{ *<non-reserved character>* | *<single quote>* |
  <control sequence> } * '
  | '{ *<non-reserved character>* | *<double quote>* |
  <control sequence> } * '

  <single quote> ::=
  ''

  <double quote> ::=
  ""

  A character string litaral of length 1, enclosed in apostrophes (e.g. 'M') may also serve as a charater literal.
- 5.2.4.9 Floating point literal
  Note: This is an extension to Z.200/1988 ans was taken from Z.200/1992.

  syntax:

  <floating point literal> ::=
  <unsigned floating point literal>
  | *<signed floating point literal>*

  <unsigned floating point literal> ::=
  <digit sequence> . [*<digit sequence>*] [*<exponent>*]
  | [*<digit sequence>*] . *<digit sequence>* [*<exponent>*]

  <signed floating point literal> ::=
  - *<unsigned floating point literal>*

  <digit sequence> ::=
  <digit> { *<digit>* | _ } *

<exponent> ::=
[E | D | e | d] <digit sequence>
| [E | D | e | d] - <digit sequence>

- 5.2.14 Start Expression
  The START expression is not implemented.
- 5.3 Values and Expressions
  The undefined value, denoted by '*', is not implemented.
- 5.3.8 Operand-5
  The *<string repetition operator>* is defined as:

  <string repetition operator> ::=
  (<integer expression>)
- 6.4 Case Action
  There may be only one case selector specified. The optional range list must not be specified.
- 6.5 Do Action
  A Do-Action without control part is not implemented. Grouping of statements can be achieved via BEGIN and END. A location enumeration is not allowed for BIT strings, only for (varying) CHAR strings and ARRAYs.

  The expression list in a DO WITH must consist of locations only.
- 6.13 Start Action
  The syntax of the START action is changed to:

  <start action> ::=
  START *<process name> (<copy number> [, <actual parameter list>])*
  [**SET** *<instance location>]*

  <copy number> ::=
  <integer expression>
- 6.16 Delay Action
  The optional PRIORITY specification need not be a constant.
- 6.17 Delay Case Action
  The optional SET branch and the, also optional, PRIORITY branch must be seperated by ';'.
- 6.18 Send Action
  The send action must define a destination instance (via the TO branch), since undirected signals are not supported. The optional PRIORITY specification need not be a constant. Additional to the data transported by the signal, there will be a user defined argument.

  The syntax of the *<send signal action>* is therefore:

  <send signal action> ::=
  SEND *<signal name> [(<value>* {, *<value>* } **)]*
  [**WITH** *<expression>]*
  TO *<instance primitive value> [<priority>]*

  The default priority can be specified by the compiler directive SEND_SIGNAL_DEFAULT_PRIORITY. If this also is omitted, the default priority is 0.

- 6.20.3 CHILL value built-in calls
  The CHILL value buit-in calls are enhanced by some calls, and other calls will have different arguments as described in Z.200/1988. Any call not mentioned here is the same as described in Z.200/1988.

  syntax:

```
CHILL value built-in routine call> ::=
            ADDR (<location>)
          | PRED (<pred succ argument>)
          | SUCC (<pred succ argument>)
          | ABS (<numeric expression>)
          | LENGTH (<length argument>)
          | SIN (<floating point expression>)
          | COS (<floating point expression>)
          | TAN (<floating point expression>)
          | ARCSIN (<floating point expression>)
          | ARCCOS (<floating point expression>)
          | ARCTAN (<floating point expression>)
          | EXP (<floating point expression>)
          | LN (<floating point expression>)
          | LOG (<floating point expression>)
          | SQRT (<floating point expression>)
          | QUEUE_LENGTH (<buffer location> | <event location>)
          | GEN_INST (<integer expression> | <process name> ,
                          <integer expression>)
          | COPY_NUMBER (<instance expression>)
          | GEN_PTYE (<process name>)
          | PROC_TYPE (<instance expression>)
          | GEN_CODE (<process name> | <signal name>)
          | DESCR (<location>)

<pred succ argument> ::=
            <discrete expression>
          | <bound reference expression>

<numeric expression> ::=
            <integer expression>
          | floating point expression>

<length argument> ::=
            <string location>
          | <string expression>
          | <string mode name>
          | <event location>
          | <event mode name>
          | <buffer location>
          | <buffer mode name>
          | <text location>
          | <text mode name>
```

semantics:

ADDR is derived syntax for `->` <*location*>.

PRED and *SUCC* delivers respectively, in case of a *discrete expression*, the next lower or higher discrete value of their argument, in case of *bound reference expression* these built-in calls deliver a pointer to the previous or next element.

ABS is defined on numeric values, i.e. integer values and floating point values, delivering the corresponding absolute value.

LENGTH is defined on

- string and text locations and string expressions, delivering the length of them;
- event locations, delivering the **event length** of the mode of the location;
- buffer locations, delivering the **buffer length** of the mode of the location;
- string mode names, delivering the **string length** of the mode;
- text mode names, delivering the **text length** of the mode;
- buffer mode names, delivering the **buffer length** of the mode;
- event mode names, delivering the **event length** of the mode;
- Additionally, *LENGTH* also may be used on the left hand side of an assignment to set a new length of a *varying character string location*. However, to avoid undefined elements in the varying string, the new length may only be less or equal to the current length. Otherwise a **RANGEFAIL** exception will be generated.

SIN delivers the sine of its argument (interpreted in radians).

COS delivers the cosine of its argument (interpreted in radians).

TAN delivers the tangent of its argument (interpreted in radians).

ARCSIN delivers the sin -1 function of its argument.

ARCCOS delivers the cos -1 function of its argument.

ARCTAN delivers the tan -1 function of its argument.

EXP delivers the exponential function, where x is the argument.

LN delivers the natural logarithm of its argument.

LOG delivers the base 10 logarithm of its argument.

SQRT delivers the sqare root of its argument.

QUEUE_LENGTH delivers either the number of sending delayed processes plus the number of messages in a buffer queue (if the argument is a *buffer location*), or the number of delayed processes (if the argument specifies an *event location*) as *integer expression*.

GEN_INST delivers an *instance expression* constructed from the arguments. Both arguments must have the *&INT*-derived class.

COPY_NUMBER delivers as *&INT*-derived class the copy number of an *instance location*.

GEN_PTYPE delivers as *&INT*-derived class the associated number of the *process name*.

PROC_TYPE delivers as *&INT*-derived class the process type of an *instance expression*.

GEN_CODE delivers as *&INT*-derived class the associated number of the *process name* or *signal name*.

DESCR delivers a *free reference expression* pointing to a structure with the following layout describing the *location* argument.

```
SYNMODE __tmp_descr = STRUCT (p PTR, l ULONG);
```

- 7.4.2 Associating an outside world object
  The syntax of the associate built-in routine call is defined as:

  <associate built-in routine call> ::=
  ASSOCIATE (<association location>, <string expression>, `[`*, <string expression>* `]` *)*

  The ASSOCIATE call has two parameters besides the association location: a pathname and an optional mode string.

  The value of the first string expression must be a pathname according to the rules of the underlying operating system. (Note that a relative pathname implies a name relative to the working directory of the process.)

  The mode string may contain the value "VARIABLE", which requests an external representation of records consisting of an UINT record length followed by as many bytes of data as indicated by the length field. Such a file with variable records is not indexable.

  A file with variable records can be written using any record mode. If the record mode is CHARS(n) VARYING, the record length is equal to the actual length of the value written. (Different record may have differing lengths.) With all other record modes, all records written using the same access mode will have the same length, but will still be prefixed with the length field. (Note that by re-connecting with different access modes, the external representation may ultimately contain records with differing lengths.)

  A file with variable records can only be read by using a record mode of CHARS(n) VARYING.

- 7.4.2 Accessing association attributes
  The value of the READABLE and WRITEABLE attributes is determined using the file status call provided by the operating system. The result will depend on the device being accessed, or on the file mode.

  The INDEXABLE attribute has the value false for files with variable records, and for files associated with devices not supporting random positioning (character devices, FIFO special files, etc.).

  The variable attribute is true for files associated with the mode sting "VARIABLE", and false otherwise.

- 7.4.5 Modifying association attributes
  The syntax of the MODIFY built-in routine call is defined as:

  <modify built-in call> ::=
  MODIFY (<association location>, <string expression>)

  At present, MODIFY accepts a character string containing a pathname in addition to the association location, which will cause a renaming of the associated file.

- 7.4.9 Data transfer operations
  READRECORD will fail (causing READFAIL) if the number of bytes from the current

position in the file to the end of the file is greater than zero but less than the size of the record mode, and no data will be transferred. (If the number of bytes is zero, no error occurs and OUTOFFILE will return TRUE.)

The number of bytes transferred by READRECORD and WRITERECORD is equal to the size of the record mode of the access location. Note that the internal representation of this mode may vary depending on the record mode being packed or not.

- 7.5 Text Input Output
  Sequential text files will be represented so as to be compatible with the standard representation of texts on the underlying operating system, where control characters are used to delimit text records on files as well as to control the movement of a cursor or printing head on a device.

  For indexed text files, records of a uniform length (i.e. the size of the text record, including the length field) are written. All i/o codes cause an i/o transfer without any carriage control characters being added to the record, which will be expanded with spaces.

  An indexed text file is therefore not compatible with the standard text representation of the underlying operating system.

- 7.5.3 Text transfer operations
  The syntax of <*text argument*> is changed to:

```
<text argument> ::=
              <text location>
            | <predefined text location>
            | <varying string location>

<predefined text location> ::=
              STDIN
            | STDOUT
            | STDERR
```

  NOTE: The identifiers STDIN, STDOUT, and STDERR are predefined. Association and connection with files or devices is done according to operating system rules.

  The effect of using READTEXT or WRITETEXT with a character string location as a text argument (i.e. the first parameter) where the same location also appears in the i/o list is undefined.

  The current implementation of formatting assumes run-to-completion semantics of CHILL tasks within an image.

- 7.5.5 Conversion
  Due to the implementation of <*floating point modes*> the syntax is changed to:

```
<conversion clause> ::=
          <conversion code> { <conversion qualifier } *
          [ <clause width> ]

<conversion code> ::=
          B | O | H | C | F

<conversion qualifier> ::=
```

L | *E* | *P*<*character*>

<*clause width*> ::=
{ <*digit*> } + | *V*
| <*real clause width*>

<*real clause width*> ::=
{ { <*digit*> + | *V* } : { { <*digit*> } + | *V* }

Note: The <*real clause width*> is only valid for <*conversion code*> 'C' or 'F'.

- 7.5.7 I/O control
  To achieve compatibility of text files written with CHILL i/o with the standard representation of text on the underlying operating system the interpretation of the i/o control clause of the format deviates from Z.200. The following table shows the i/o codes together with the control characters written before and after the text record, to achieve the indicated function:

  '/' Write next record (record, line feed)

  '+' Write record on next page (form feed, record, line feed)

  '-' Write record on current line (record, carriage return)

  '?' Write record as a prompt (carriage return, record)

  '!' Emit record (record).

  '=' Force new page for the next line: The control character written before the next record will be form feed, irrespective of the i/o control used for transferring the record.

  When reading a text file containing control characters other than line feed, these characters have to be reckoned with by the format used to read the text records.
- 11.2.2 Regionality
  Regionality is not implemented at all, so there is no difference in the generated code when REGION is substituted by MODULE in a GNUCHILL compilation unit.
- 11.5 Signal definition statement
  The <*signal definition statement*> may only occur at module level.
- 12.3 Case Selection
  The syntax of <*case label specification*> is changed to:

  <*case label specification*> ::=
  (<*case label*> {, <*case label*> } *)

  <*case label*> ::=
  <*discrete literal expression*>
  | <*literal range*>
  | <*discrete mode name*>
  | **ELSE**

7 Compiler Directives

- ALL_STATIC_ON, ALL_STATIC_OFF
  These directives control where procedure local variables are allocated. ALL_STATIC_ON turns allocation of procedure local variables in the data space ON, regardless of the keyword STATIC being used or not. ALL_STATIC_OFF places procedure local variables in the stack space. The default is ALL_STATIC_OFF.
- RANGE_ON, RANGE_OFF
  Turns generation of rangecheck code ON and OFF.
- USE_SEIZE_FILE <character string literal>
  Specify the filename (as a character string literal) where subsequent SEIZE statements are related to. This directive and the subsequent SEIZEs are written to a possibly generated grant file for this module.

  ```
  <> USE_SEIZE_FILE "foo.grt" <>
  SEIZE bar;
  ```
- USE_SEIZE_FILE_RESTRICTED "filename"
  Same as USE_SEIZE_FILE. The difference is that this directive and subsequent SEIZEs are *not* written to a possibly generated grant file.
- PROCESS_TYPE = <integer expression>
  Set start value for all PROCESS delclarations. This value automatically gets incremented after each PROCESS declaration and may be changed with a new PROCESS_TYPE compiler directive.
- SIGNAL_CODE = <integer expression>
  Set start value for all SIGNAL definitions. This value automatically gets incremented after each SIGNAL definition and may be changed with a new SIGNAL_CODE compiler directive.
- SEND_SIGNAL_DEFAULT_PRIORITY = <integer expression>
  Set default priority for send signal action.
- SEND_BUFFER_DEFAULT_PRIORITY = <integer expression>
  Set default priority for send buffer action.

  Note: Every <integer expression> in the above mentioned compiler directives may also be specified by a SYNONYM of an integer type.

  ```
  SYN first_signal_code = 10;
  <> SIGNAL_CODE = first_signal_code <>
  SIGNAL s1;
  ```

8 Language Definition References

- CCITT High Level Language (CHILL) Recommendation Z.200 ISO/IEC 9496, Geneva 1989 ISBN 92-61-03801-8
- An Analytic Description of CHILL, the CCITT high-level language, Branquart, Louis & Wodon, Springer-Verlag 1981 ISBN 3-540-11196-4
- CHILL User's Manual CCITT, Geneva 1986 ISBN 92-61-02601-X
- Introduction to CHILL CCITT, Geneva 1983 ISBN 92-61-017771-1
- CHILL CCITT High Level Language Proceedings of the 5th CHILL Conference North-Holland, 1991 ISBN 0 444 88904 3
- Introduction to the CHILL programming Language TELEBRAS, Campinas, Brazil 1990

Z.200 is mostly a language-lawyer's document, but more readable than most. The User's Guide is more readable by far, but doesn't cover the whole language. Our copies of these documents came through Global Engineering Documents, in Irvine, CA, USA. (714)261-1455.

Table of Contents