1 SUMMARY

This package defines a derived type capable of supporting a variety of sparse matrix storage schemes. Its principal use is to allow exchange of data between HSL subprograms and other codes.

ATTRIBUTES — **Version:** 1.0.0. **Types:** `HSL_ZD01_single`, `HSL_ZD01_double`, `HSL_ZD01_integer`, `HSL_ZD01_complex`, `HSL_ZD01_double_complex`. **Remark:** This package is also included in the HSL Archive. **Original date:** March 2000. **Origin:** N. I. M. Gould and J. K. Reid, Rutherford Appleton Laboratory. **Language:** Fortran 95 or Fortran 90. **Licence:** A third-party licence for this package is available without charge.

2 HOW TO USE THE PACKAGE

Access to the package requires a `USE` statement such as

Single precision version

```
USE HSL_ZD01_single
```

Double precision version

```
USE HSL_ZD01_double
```

Integer version

```
USE HSL_ZD01_integer
```

Complex version

```
USE HSL_ZD01_complex
```

Double complex version

```
USE HSL_ZD01_double_complex
```

If it is required to use two modules at the same time, the derived type `ZD01_type` (Section 2.1) must be renamed in one of the `USE` statements.

2.1 The derived data type

A single derived data type, `ZD01_type`, is accessible from the package. It is intended that, for any particular application, only those components which are needed will be set. In the Fortran 95 code, all the pointer compoents are initialized to be null. The components are:

`id` is a pointer array of rank one and type default `CHARACTER(1)` that may be used to identify the matrix.

`type` is a pointer array of rank one and type default `CHARACTER(1)` that may be used to indicate the properties of the matrix in question.

`m` is a scalar component of type default `INTEGER` that may be used to hold the number of rows in the matrix.

`n` is a scalar component of type default `INTEGER` that may be used to hold the number of columns in the matrix.

`ne` is a scalar component of type default `INTEGER` that may be used to hold the number of entries in the matrix.

`row` is a pointer array of rank one and type default `INTEGER` that may be used to hold the row indices of the entries of the matrix.

`col` is a pointer array of rank one and type default `INTEGER` that may be used to hold the column indices of the entries of the matrix.

`val` is a pointer array of rank one and type default `REAL` (double precision `REAL` in `HSL_ZD01_double`, `COMPLEX` in `HSL_ZD01_complex`, `COMPLEX(KIND(0.0D0)` in `HSL_ZD01_double_complex`, `INTEGER` in

HSL_ZD01_integer) that may be used to hold the numerical values of the entries of the matrix.

ptr is a pointer array of rank one and type default INTEGER that may be used to hold the starting positions of each row in a row-wise storage scheme, or the starting positions of each column in a column-wise storage scheme.

2.2 Conversions between character variables and character arrays

To assist use of the character arrays in the components id and type, the module provides two procedures:

ZD01_put is a subroutine that allocates a character array and sets its components from a character variable.

ZD01_get is a function that obtains the elements of a character array as a character variable.

Allocate a character array and set its components

```
CALL ZD01_put(array,string,stat)
```

array is a rank-one pointer array of type CHARACTER(LEN=1). If string is present, array is allocated with size LEN_TRIM(string) and its elements are given the values string(i:i), i = 1, 2, ... ; otherwise, array is allocated to be of size zero.

string is an optional scalar of INTENT(IN) and type CHARACTER with any character length.

stat is an optional scalar of INTENT(OUT) and type default INTEGER. If present, an ALLOCATE statement with this as its STAT= variable is executed and a successful allocation will be indicated by the value zero. If absent, an ALLOCATE statement without a STAT= variable is executed.

Obtain the elements of a character array as a character variable

```
string = ZD01_get(array)
```

array is a rank-one array of type CHARACTER(LEN=1). It is not altered.

The result is scalar and of type CHARACTER(LEN=SIZE(array)). ZD01_get(i:i) is given the value array(i), i = 1, 2, ..., SIZE(array).

3 GENERAL INFORMATION

Other modules used directly: The procedures of Section 2.2 are in the module HSL_ZD01_char, which is used by HSL_ZD01_single, HSL_ZD01_double, HSL_ZD01_complex, HSL_ZD01_double_complex, and HSL_ZD01_integer.

Input/output: None.

Restrictions: None.

5 EXAMPLE OF USE

The following code stores the upper-triangular part of the symmetric matrix

$$\begin{pmatrix} 1.0 & & \\ & & 1.0 \\ & 1.0 & \end{pmatrix},$$

whose identifier is “Sparse”, using a coordinate sparse matrix storage format. It writes out details of the stored data and then deallocates the pointer components.

```
PROGRAM MAIN
   USE HSL_ZD01_double
   INTEGER :: i
   TYPE ( ZD01_type ) :: A
   A%n = 3 ; A%ne = 2

   ALLOCATE( A%row( A%ne ), A%col( A%ne ), A%val( A%ne ) )
   CALL ZD01_put  (A%id,'Sparse')
   CALL ZD01_put  (A%type)
   A%row( 1 ) = 1 ; A%col( 1 ) = 1 ; A%val( 1 ) = 1.0
   A%row( 2 ) = 2 ; A%col( 2 ) = 3 ; A%val( 2 ) = 1.0

   WRITE( 6, "( 3A, I2, //, A )" ) ' Matrix ', ZD01_get(A%id), &
          ' dimension', A%n, ' row col  value '
   DO i = 1, A%ne
      WRITE( 6, "( I3, 1X, I3, ES9.1 )" ) A%row( i ), A%col( i ), A%val( i )
   END DO
   DEALLOCATE( A%id, A%row, A%col, A%val)
END PROGRAM MAIN
```

This produces the following output:

```
 Matrix Sparse dimension 3

 row col  value
  1   1  1.0E+00
  2   3  1.0E+00
```