/STDS=*dupsReplacedWave*

Specifies the output wave generated by /ST. If you omit /STDS then the output wave created by /ST is T_ReplacedDuplicates in the current data folder. /STDS without /ST has no effect.

Details

FindDuplicates scans *srcWave* and identifies duplicate values. The first instance of any value is not considered a duplicate. Duplicates are either identical, as is the case with integer or text waves, or values that are within a specified tolerance in the case of single-precision or double-precision numeric waves.

Text comparison is case-sensitive unless you use /CI or /CI/LOC.

The operation creates wave references for the waves specified by the various flags above. See **Automatic Creation of WAVE References** on page IV-72 for details.

Example

```
Function DemoFindDuplicates(mode)
   int mode         // 0=case sensitive; 1=/CI; 2=/CI/LOC

   Make/O/T sourceText={"A","a", "Å","å", "B","b"}
   switch(mode)
      case 0:       // Case sensitive
         // Returns {"A","a","Å","å","B","b"}
         FindDuplicates/FREE/RT=output sourceText
         break
      case 1:       // Case insensitive for ASCII only
         // Returns {"A","Å","å","B"}
         FindDuplicates/FREE/RT=output/CI sourceText
         break
      case 2:       // Case insensitive, locale aware
         // Returns {"A","Å","B"}
         FindDuplicates/FREE/RT=output/CI/LOC sourceText
         break
   endswitch

   Print sourceText
   Print output
End
```

See Also

FindLevels, **FindValue**, **Sort**, **TextHistogram**

FindLevel

***FindLevel** [flags] waveName, level*

The FindLevel operation searches the named wave to find the X value at which the specified Y *level* is crossed.

Flags

/B=*box*

Sets box size for sliding average. If /B=*box* is omitted or *box* equals 1, no averaging is done. If you specify an even box size then the next higher (odd) integer is used. If you use a box size greater than 1, FindLevel will be unable to find a level crossing that occurs in the first or last (*box*-1)/2 points of the wave since these points don't have enough neighbors for computing the derived average wave values.

/EDGE=*e*

Specifies searches for either increasing or decreasing level crossing.

e=1:	Searches only for crossing where Y values are increasing as *level* is crossed from wave start towards wave end.
e=2:	Searches only for crossing where the Y values are decreasing as *level* is crossed from wave start towards wave end.
e=0:	Same as no /EDGE flag (searches for either increasing and decreasing level crossing).

/P	Computes the X crossing location in terms of point number. If /P is omitted, the level crossing location is computed in terms of X values.
/Q	Don't print results in history and don't report error if *level* is not found.
/R=(*startX*,*endX*)	Specifies an X range of the wave to search. You may exchange *startX* and *endX* to reverse the search direction.
/R=[*startP*,*endP*]	Specifies a point range of the wave to search. You may exchange *startP* and *endP* to reverse the search direction. If you specify the range as /R=[*startP*] then the end of the range is taken as the end of the wave. If /R is omitted, the entire wave is searched.
/T=*dx*	Search for two level crossings. *dx* must be less than *minWidthX*, so you must also specify /M if you use /T. (FindLevel limits *dx* so that second search start isn't beyond where the first search for next edge will be.)
/T=*dx*	Performs a second search after finding the initial level crossing. The second search starts *dx* units beyond the initial level crossing and looks back in the direction of the initial crossing. If FindLevel finds a second level crossing, it sets V_LevelX to the average of the initial and second crossings. Otherwise, it sets V_LevelX to the initial crossing.

Details

FindLevel scans through the wave comparing *level* to values derived from the Y values of the wave. Each derived value is a sliding average of the Y values.

FindLevel searches for two derived wave values that straddle *level*. If it finds these values it computes the X value at which *level* is located by linearly interpolating between the straddling Y values.

FindLevel does not locate values exactly equal to *level*; it locates transitions through *level*. See **BinarySearch** for one method of locating exact values.

FindLevel reports its results by setting these variables:

`V_flag`	0: *level* was found. 1: *level* was not found.
`V_LevelX`	Interpolated X value at which *level* was found, or the corresponding point number if /P is specified.
`V_rising`	0: Y values at the crossing are decreasing from wave start towards wave end. 1: Y values at the crossing are increasing.

If you omit the /Q flag then FindLevel also reports its results by printing them in the history area.

If *level* is not found, and if you omit the /Q flag, FindLevel generates an error which puts up an error alert and halts execution of any command line or macro that is in progress.

V_LevelX is returned in terms of the X scaling of the named wave unless you use the /P flag, in which case it is in terms of point number.

FindLevel Handling of NaNs

In Igor Pro 8.00 and later, FindLevel handles NaN values differently than previous versions. Now if level falls between two non-NaN wave Y values with NaNs between them, those two Y values and their associated X scaling values are used to linearly interpolate the X location of the level crossing. Igor7 and earlier fail to find a crossing and set V_LevelX to NaN.

For example:

```
// Prints 2.5 in Igor8 or later, NaN in Igor7 or before
Make/O wave0 = {0, 1, NaN, NaN, 4, 5}
FindLevel/Q wave0, 2.5; Print V_levelX
```

You can revert to the pre-Igor 8 behavior by executing:

```
SetIgorOption UseIP6FindLevel = 1
```