ctools - Bug #631

ctselect: Specifying RA=DEC=0 leads to use of pointing for ROI centre

12/14/2012 02:19 PM - Knödlseder Jürgen

Status:	Closed	**Start date:**	12/14/2012
Priority:	Normal	**Due date:**	
Assigned To:	Knödlseder Jürgen	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	00-06-00		

Description

If the arguments ra and dec are both set to 0.0, the pointing is used instead of the ra and dec parameters for the ROI centre. This prevents using ra=0 and dec=0 as selection parameters.

I recall that I used this feature in the past for automatically selecting the pointing direction as the ROI centre, so this is more a feature than a bug. However, it's obvious that ra=0 and dec=0 is a valid choice for the ctselect parameters, and using these parameters should not use to a specific behaviour.

We should introduce a new hidden parameter usepointing that explicitly tells ctselect to use the pointing as ROI centre.

History

#1 - 12/14/2012 03:49 PM - Knödlseder Jürgen

- % Done changed from 0 to 10

Note that the code documentation says

@todo Use INDEF instead of 0.0 for pointing as RA/DEC selection

This seems another interesting solution, but requires first the implementation of the INDEF logic in the GPar class. Needs some further reading on the IRAP parameter file interface.

Here the relevant code in ape_util.c that shows how APE (HEASARC) is handling the thing. Apparently, the string2double() and string2long() functions return a status that can be undefined for INDEF and NONE (case insensitive) or NaN/LONG_MAX for INF, INFINITY or NAN:

```
/* Strictly convert input string to output bool, ignoring leading/trailing white space, and flagging every possible error. */
static int string2bool(const char * input, char * result) {
  int status = eOK;
  if (0 != result) {
    *result = 0;
    if (0 != input) {
      /* Truncate white space from input string. */
      char * copy = truncate_white_space(input);

      /* Compare input string (case insensitive) to various boolean expressions. */
      if (0 == ci_strcmp(copy, "true") || 0 == ci_strcmp(copy, "t") ||
        0 == ci_strcmp(copy, "yes") || 0 == ci_strcmp(copy, "y")) {
        *result = 1;
      } else if (0 == ci_strcmp(copy, "false") || 0 == ci_strcmp(copy, "f") ||
        0 == ci_strcmp(copy, "no") || 0 == ci_strcmp(copy, "n")) {
        *result = 0;
      } else {
        /* String does not contain a valid bool expression. */
        status = eConversionError;
      }

      /* Clean up. */
      free(copy); copy = 0;
    } else {
      status = eNullPointer;
    }
  } else {
```

```
    status = eNullPointer;
  }
  return status;
}

/* Strictly convert input string to output double, ignoring leading/trailing white space, and flagging every possible error. */
static int string2double(const char * input, double * result) {
  int status = eOK;
  if (0 != result) {
    *result = 0.;
    if (0 != input) {
      char * remainder = 0;
      /* Skip leading white space. */
      while (0 != isspace(*input)) ++input;

      if ('\0' == *input || 0 == ci_strcmp(input, "indef") || 0 == ci_strcmp(input, "none") || 0 == ci_strcmp(input, "undef") ||
        0 == ci_strcmp(input, "undefined")) {
        status = eUndefinedValue;
      } else if (0 == ci_strcmp(input, "inf") || 0 == ci_strcmp(input, "infinity") || 0 == ci_strcmp(input, "nan")) {
        status = eNan;
      } else {
        errno = 0;
        *result = strtod(input, &remainder);
        if (ERANGE == errno) {
          if (0. == *result) status = eUnderflow;
          else status = eOverflow;
        } else if (0 != remainder && '\0' != *remainder) {
          /* Ignore trailing whitespace; the C standard is not specific about whether it's returned or eaten by the conversion. */
          while (0 != isspace(*remainder)) ++remainder;
          if ('\0' != *remainder) status = eStringRemainder;
          /* else conversion succeeded. */
        } else {
          /* Conversion succeeded! */
        }
      }
    } else {
      status = eNullPointer;
    }
  } else {
    status = eNullPointer;
  }
  return status;
}

/* Strictly convert input string to output long, ignoring leading/trailing white space, and flagging every possible error. */
static int string2long(const char * input, long * result) {
  int status = eOK;
  if (0 != result) {
    *result = 0;
    if (0 != input) {
      char * remainder = 0;
      errno = 0;
      /* Skip leading white space. */
      while (0 != isspace(*input)) ++input;

      if ('\0' == *input || 0 == ci_strcmp(input, "indef") || 0 == ci_strcmp(input, "none") || 0 == ci_strcmp(input, "undef") ||
        0 == ci_strcmp(input, "undefined")) {
        status = eUndefinedValue;
      } else if (0 == ci_strcmp(input, "inf") || 0 == ci_strcmp(input, "infinity") || 0 == ci_strcmp(input, "nan")) {
        *result = LONG_MAX;
      } else {
        *result = strtol(input, &remainder, 0);
        if (ERANGE == errno) {
          if (LONG_MIN == *result) status = eUnderflow;
          else if (LONG_MAX == *result) status = eOverflow;
          else status = eConversionError;
        } else if (0 != remainder && '\0' != *remainder) {
          /* Ignore trailing whitespace; the C standard is not specific about whether it's returned or eaten by the conversion. */
          while (0 != isspace(*remainder)) ++remainder;
          if ('\0' != *remainder) status = eStringRemainder;
          /* else conversion succeeded. */
        } else {
          /* Conversion succeeded! */
        }
      }
```

```
  } else {
    status = eNullPointer;
  }
 } else {
  status = eNullPointer;
 }
 return status;
}
```

#2 - 11/19/2013 03:23 PM - Knödlseder Jürgen

- Status changed from New to Closed

- Assigned To set to Knödlseder Jürgen

- % Done changed from 10 to 100

Parameter usepnt has been added.