Subject: Re: CParser do not check for invalid strings that spam lines
Posted by mingodad on Sun, 13 Apr 2014 07:42:48 GMT
View Forum Message <> Reply to Message

The cast to byte solved it, but I'm not sure if return to ignoring the check end of string by default is a good default !

And continuing working with witz why not register handler with a class method ?

This is something I did for FLTK:

```
class Fl_AnyClass {};

/** Default member callback type definition for all fltk widgets (by far the most used) */
typedef void (Fl_AnyClass::*Fl_MCallback)(Fl_Widget*, void*);
/** Default member callback type pointer definition for all fltk widgets */
typedef Fl_MCallback* Fl_MCallback_p; // needed for BORLAND
/** Zero parameter member callback type definition passing only the widget */
typedef void (Fl_AnyClass::*Fl_MCallback0)();
/** One parameter member callback type definition passing only the widget */
typedef void (Fl_AnyClass::*Fl_MCallback1)(Fl_Widget*);
/** Member callback type definition passing the widget and a long data value */
typedef void (Fl_AnyClass::*Fl_MCallback2)(Fl_Widget*, long);
/** Member callback type definition passing the widget and a long data value */
typedef void (Fl_AnyClass::*Fl_MCallback3)(Fl_Widget*, double);

#define MCALLBACK(wdg, mf) (wdg)->mcallback((Fl_AnyClass*)this, (Fl_MCallback)&mf)
#define THISMBACK(wdg, mf) (wdg)->mcallback((Fl_AnyClass*)this,
(Fl_MCallback)&THISCLASS::mf)

...
class FL_EXPORT Fl_Widget : public Fl_Rectangle {
  friend class Fl_Group;

  Fl_Group* parent_;
  union {
      Fl_Callback* callback_;
      Fl_MCallback mcallback_;
  };
  void* user_data_;
  //int x_,y_,w_,h_;  //Using Fl_Rectangle
  Fl_AnyClass *any_class_mcb_;
...
 /** class that holds a method for callbacks*/
  Fl_AnyClass *any_class_mcb() {return any_class_mcb_;};

  /** Gets the current class method callback function for the widget.
```

```
    Each widget has a single class method callback.
    \return current mcallback
 */
Fl_MCallback mcallback() const {return mcallback_;}

/** Sets the current class method callback function for the widget.
    Each widget has a single class method callback.
    \param[in] cb new class method callback
    \param[in] p user data
 */
void mcallback(Fl_AnyClass *klass, Fl_MCallback cb, void* p) {
    any_class_mcb_ = klass;
    mcallback_=cb;
    user_data_=p;
}

/** Sets the current class method callback function for the widget.
    Each widget has a single callback.
    \param[in] cb new method callback
 */
void mcallback(Fl_AnyClass *klass, Fl_MCallback cb) {
    any_class_mcb_ = klass;
    mcallback_=cb;
}

/** Sets the current callback method function for the widget.
    Each widget has a single method callback.
    \param[in] cb new callback
 */
void mcallback(Fl_AnyClass *klass, Fl_MCallback0 cb) {
      any_class_mcb_ = klass;
      mcallback_=(Fl_MCallback)cb;
}

/** Sets the current callback method function for the widget.
    Each widget has a single method callback.
    \param[in] cb new callback
 */
void mcallback(Fl_AnyClass *klass, Fl_MCallback1 cb) {
    any_class_mcb_ = klass;
    mcallback_=(Fl_MCallback)cb;
}

/** Sets the current callback method function for the widget.
    Each widget has a single method callback.
    \param[in] cb new callback
    \param[in] p user data
 */
```

```
  void mcallback(Fl_AnyClass *klass, Fl_MCallback2 cb, long p=0) {
     any_class_mcb_ = klass;
     mcallback_=(Fl_MCallback)cb;
     user_data_=(void*)p;
  }
...
/** Calls the widget callback.

  Causes a widget to invoke its callback function with arbitrary arguments.

  \param[in] o call the callback with \p o as the widget argument
  \param[in] arg use \p arg as the user data argument
  \see callback()
*/
void
Fl_Widget::do_callback(Fl_Widget* o,void* arg) {
  Fl_Widget_Tracker wp(this);
  if(any_class_mcb_) (*any_class_mcb_.*mcallback_)(o,arg);
  else callback_(o,arg);
  if (wp.deleted()) return;
  if (callback_ != default_callback)
    clear_changed();
}
```