1. Introduction. [LDF 2006.10.18.]

2. Copyright and licenses. [LDF 2006.10.23.]

The IWF Metadata Harvester is a package for metadata harvesting.

The author is Laurence D. Finston.

The IWF Metadata Harvester is available for downloading from the following FTP server:
`ftp://ftp.gwdg.de/pub/gnu2/iwfmdh/`

Please send bug reports to `lfinsto1@gwdg.de`

The author can be contacted at:

Laurence D. Finston
Kreuzbergring 41
D-37075 Göttingen
Germany

Email: `lfinsto1@gwdg.de`
`s246794@stud.uni-goettingen.de`

3. Formatting commands. [LDF 2006.10.18.]
This section contains formatting commands. They don't appear in the TEX output of CWEAVE.

4. File Lists. [LDF 2006.10.18.]

5. Source Files. [LDF 2006.10.18.]

6. Logical and Hierarchical Order. [LDF 2006.10.18.]

⟨ `nonwin.web` 9 ⟩ .... Main header file when using The GNU Compiler Collection
⟨ `stdafx.web` 18 ⟩ .... Main header file when using Microsoft Visual Studio
⟨ `querytyp.web` 45 ⟩ .... **Query_Type**
⟨ `qtgensql.web` 120 ⟩ .... **Query_Type** :: *generate_sql_string* definition
⟨ `dtsrctyp.web` 155 ⟩ .... **Datasource_Type**
⟨ `dttmtype.web` 25 ⟩ .... **Date_Time_Type**
⟨ `idtype.web` 184 ⟩ .... **Id_Type**
⟨ `scnrtype.web` 209 ⟩ .... **Scanner_Type**
⟨ `scanner.web` 245 ⟩ .... Type declarations and the *yylex* function
⟨ `scantest.web` 696 ⟩ .... The *_tmain* function when using Microsoft Visual Studio
⟨ `parser.w` 359 ⟩ .... The parser
⟨ `main.web` 701 ⟩ .... The *main* function when using The GNU Compiler Collection

7. Parser files. [LDF 2006.11.16.]

Log

[2006.11.16.] Added this section.

⟨ `parser.w` 359 ⟩ .... The main parser input file.
⟨ `grpstmnt.w` 398 ⟩ .... Group statements.
⟨ `declrtns.w` 401 ⟩ .... Declarations.
⟨ `variabls.w` 422 ⟩ .... Variables.
⟨ `assign.w` 446 ⟩ .... Assignment.
⟨ `commands.w` 565 ⟩ .... Commands.
⟨ `dttmexp.w` 625 ⟩ .... Datetime expressions.
⟨ `queryexp.w` 579 ⟩ .... Query expressions.
⟨ `dtsrcexp.w` 603 ⟩ .... Datasource expressions.

8. Alphabetical Order. [LDF 2006.09.27.]

9. Non-Windows Code (nonwin.web). [LDF 2006.10.20.]

Log

[LDF 2006.10.20.] Created this file.

[LDF 2006.11.03.] Now including `unistd.h` in order to be be able to use *getopt*.

⟨ nonwin.web 9 ⟩ ≡

```
static char id_string[] = "$Id: nonwin.web,v 1.3 2007/01/02 08:35:42 lfinsto1 Exp $";
```

This code is cited in sections 6, 8, and 17.

This code is used in section 17.

10. Include files.

⟨ Include files 10 ⟩ ≡

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <cmath>
#include <bitset>
#include <ios>
#include <iomanip>
#include <iostream>
#include <iosfwd>
#include <fstream>
#include <sstream>
#if 0   /* 1 */
#include <strstream>
#endif
#include <new>
#include <ctype.h>
#include <string.h>
#include <map>
#include <stack>
#include <vector>
#include <time.h>
```

See also sections 20, 26, 46, 121, 156, 185, 210, 246, 360, 646, 697, and 702.

This code is used in sections 16, 24, 43, 118, 154, 182, 207, 243, 357, 643, 695, 700, and 715.

11. Type declarations. [LDF 2006.10.20.]

Log

[LDF 2006.10.20.] Added this section.

---

12. Mutex_Type. [LDF 2006.10.20.]

Log

[LDF 2006.10.20.] Added this type declaration.

⟨ **Mutex_Type** declaration 12 ⟩ ≡

```
struct Mutex_Type {
  int Lock(void)
  {
    return 0;
  }
  ;
  int Unlock(void)
  {
    return 0;
  }
  ;
};
```

This code is used in section 16.

13. Forward declarations. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

⟨ Forward declarations 13 ⟩ ≡

```
class Query_Type;
typedef Query_Type *Query_Node;
class Scanner_Type;
typedef Scanner_Type *Scanner_Node;
```

See also sections 28, 48, 158, 187, 212, and 248.

This code is used in sections 16, 43, 44, 118, 119, 182, 183, 207, 208, 243, 244, 357, and 358.

14. extern declarations for global variables. [LDF 2006.10.19.]

Log

[LDF 2006.10.19.] Added this section.

[LDF 2006.11.23.] Added **extern** declarations of **ofstream** *tex_file_strm*, **Mutex_Type** *tex_mutex*, **unsigned short** *tex_file_ctr*, and **string** *tex_filename_str*.

[LDF 2006.11.30.] Added **extern** declaration of **const string** *copyright_tex_str*.

[LDF 2006.11.30.] Added **extern** declaration of **Mutex_Type** *time_mutex*.

⟨ **extern** declarations for global variables 14 ⟩ ≡

```
extern Mutex_Type cerr_mutex;
extern Mutex_Type cout_mutex;
extern Mutex_Type time_mutex;
extern ofstream tex_file_strm;
extern Mutex_Type tex_mutex;
extern unsigned short tex_file_ctr;
extern string tex_filename_str;
extern string copyright_tex_str;
extern int yydebug;
```

See also section 21.

This code is used in sections 16 and 24.

15. Putting Non-Win together. [LDF 2006.10.20.]

16. This is what's written to `nonwin.h`.

⟨ `nonwin.hh` 16 ⟩ ≡
  ⟨ Include files 10 ⟩
  using namespace std;
  ⟨ **Mutex_Type** declaration 12 ⟩
  ⟨ Forward declarations 13 ⟩
  ⟨ **extern** declarations for global variables 14 ⟩

17. This is what *isn't* compiled. The file `nonwin.c` isn't compiled, but including ⟨ `nonwin.web` 9 ⟩ here prevents CWEAVE from issuing a warning. [LDF 2006.10.24.]

⟨ `nonwin.web` 9 ⟩

18. stdafx (`stdafx.web`). [LDF 2006.10.17.]

Log

[LDF 2006.10.17.] Created this file.

⟨ `stdafx.web` 18 ⟩ ≡

```
static char id_string[] = "$Id:␣stdafx.web,v␣1.3␣2007/01/02␣12:40:50␣lfinsto1␣Exp␣$";
```

This code is cited in sections 6 and 8.

This code is used in section 23.

19. Preprocessor macro calls. [LDF 2006.10.17.]

⟨ Preprocessor macro calls 19 ⟩ ≡

```
#ifdef WIN_LDF
#pragma once
#endif
```

See also sections 27, 47, 157, 186, 211, 247, and 647.

This code is used in sections 24, 44, 119, 183, 208, 244, and 358.

20. Include files in `stdafx.h`. [LDF 2006.10.17.]

⟨ Include files 10 ⟩ +≡

```
#include <iostream>
#include <tchar.h>
#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS   /* einige CString-Konstruktoren sind explizit */
#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN   /* Selten verwendete Teile der Windows-Header nicht einbinden */
#endif
#include <afx.h>
#include <afxwin.h>   /* MFC-Kern- und Standardkomponenten */
#include <afxext.h>   /* MFC-Erweiterungen */
#include <afxdtctl.h>
    /* MFC-Untersttzung fr allgemeine Steuerelemente von Internet Explorer 4 */
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>   /* MFC-Untersttzung fr allgemeine Windows-Steuerelemente */
#endif   /* _AFX_NO_AFXCMN_SUPPORT */
#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS   /* einige CString-Konstruktoren sind explizit */
#include <atlbase.h>
#include <stdio.h>
#include <stdlib.h>
#include <ios>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <strstream>
#include <new>
#include <ctype.h>
#include <string.h>
```

```
#include <map>
#include <stack>
#include <vector>
#include <time.h>
#include <afxmt.h>
#include "scanner.h"
```

21. extern declarations for global variables. [LDF 2006.10.19.]

Log

[LDF 2006.10.19.] Added this section.

⟨ **extern** declarations for global variables 14 ⟩ +≡
  extern ofstream *log_strm*;
  extern CMutex *log_strm_mutex*;
  extern CMutex *cerr_mutex*;
  extern CMutex *cout_mutex*;
  extern ifstream *in_strm*;

22. Putting stdafx.web together.

23. This is what's compiled. [LDF Undated.]

```
#include "stdafx.h"
  ⟨ stdafx.web 18 ⟩
```

24. This is what's written to `stdafx.h`.

⟨ `stdafx.hh` 24 ⟩ ≡
  ⟨ Preprocessor macro calls 19 ⟩
  ⟨ Include files 10 ⟩
  ⟨ **extern** declarations for global variables 14 ⟩

25. Date_Time_Type (`dttmtype.web`). [LDF 2006.10.17.]

Log

[LDF 2006.10.31.] Created this file.

⟨ `dttmtype.web` 25 ⟩ ≡
  static char *id_string*[ ] = `"$Id:␣dttmtype.web,v␣1.4␣2007/01/02␣12:39:42␣lfinsto1␣Exp␣$"`;

This code is cited in sections 6 and 8.

This code is used in section 43.

26. Include files.

⟨ Include files 10 ⟩ +≡
#include `"localldf.h"`
#ifdef `NON_WIN_LDF`
#include `"nonwin.h"`
#else
#include `"stdafx.h"`
#endif
#include `"parser.hxx"`

27. Preprocessor macro calls.

⟨ Preprocessor macro calls 19 ⟩ +≡
#ifdef `WIN_LDF`
#pragma *once*
#endif

28. Forward declarations. [LDF 2006.10.31.]

⟨ Forward declarations 13 ⟩ +≡

```
class Query_Type;
class Date_Time_Type;
typedef Date_Time_Type *Date_Time_Node;
namespace Scan_Parse {
  int datetime_assignment_func_0(void *v, Date_Time_Node &curr_date_time_node, int specifier, int
      op, void *val, int type);
  int datetime_assignment_func_1(void *v, Date_Time_Node &d, int op, void *&vec);
};
```

29. class Date_Time_Type declaration. [LDF 2006.10.17.]

Log

!! BUG FIX: Made function declarations **public**.

[LDF 2006.12.18.] Added **friend** declaration for *yyparse*.

[LDF 2006.12.18.] Added **friend** declaration for **Scan_Parse** :: *datetime_assignment_func_0*.

[LDF 2006.12.18.] Added **friend** declaration for **Scan_Parse** :: *datetime_assignment_func_1*.

⟨ Declare **class Date_Time_Type** 29 ⟩ ≡

```
class Date_Time_Type {
  friend class Query_Type;
  friend int Scan_Parse::datetime_assignment_func_0(void *v, Date_Time_Node
      &curr_date_time_node, int specifier, int op, void *val, int type);
  friend int Scan_Parse::datetime_assignment_func_1(void *v, Date_Time_Node &d, int op, void
      *&vec);
  friend int yyparse(void *);
  friend ostream &operator<<(ostream & o, Date_Time_Type &d);
  short *year_range_begin;
  short *year_range_end;
  short *year;
  unsigned short *month;
  unsigned short *day;
  unsigned short *hour;
  unsigned short *minute;
  float *second;
public: ⟨ Declare Date_Time_Type functions 32 ⟩
};
```

This code is used in sections 43 and 44.

30. Date_Time_Type functions. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

---

31. Constructors and setting functions. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

---

32. Default constructor. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this function.

⟨ Declare **Date_Time_Type** functions 32 ⟩ ≡
  Date_Time_Type(**void**);

See also sections 34, 36, and 40.

This code is used in section 29.

33.

⟨ Define **Date_Time_Type** functions 33 ⟩ ≡
  Date_Time_Type :: **Date_Time_Type**(**void**)
  {
    year_range_begin = 0;
    year_range_end = 0;
    year = 0;
    month = 0;
    day = 0;
    hour = 0;
    minute = 0;
    second = 0;
    return;
  } /* End of the default **Date_Time_Type** constructor definition. */

See also sections 35, 37, and 41.

This code is used in section 43.

34. Destructor. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this function.

⟨ Declare **Date_Time_Type** functions 32 ⟩ +≡
  ~**Date_Time_Type**(**void**);

35.

⟨ Define **Date_Time_Type** functions 33 ⟩ +≡

```
Date_Time_Type::~Date_Time_Type(void)
{
  delete year_range_begin;
  delete year_range_end;
  delete year;
  delete month;
  delete day;
  delete hour;
  delete minute;
  delete second;
  return;
} /* End of the default Date_Time_Type destructor definition. */
```

36. Assignment. [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this function.

⟨ Declare **Date_Time_Type** functions 32 ⟩ +≡

```
Date_Time_Node operator=(const Date_Time_Type &d);
```

37.

```
⟨ Define Date_Time_Type functions 33 ⟩ +≡
  Date_Time_Node Date_Time_Type::operator=(const Date_Time_Type &d)
  {
#if 0    /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣‘Date_Time_Type’␣assignment␣operator." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    temp_strm.str("");
#endif
    if (this ≡ &d) {
#ifdef DEBUG_OUTPUT
      temp_strm << "In␣‘Date_Time_Type::operator=(const␣Date_Time_Type&)’:" << endl <<
          "Self-assignment.␣␣Returning␣‘this’." << endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      temp_strm.str("");
#endif
      return this;
    }
    if (year_range_begin ∧ d.year_range_begin) {
      *year_range_begin = *d.year_range_begin;
    }
    else if (year_range_begin ∧ ¬d.year_range_begin) {
      delete year_range_begin;
      year_range_begin = 0;
    }
    else if (¬year_range_begin ∧ d.year_range_begin) {
      year_range_begin = new short(*d.year_range_begin);
    }
    if (year_range_end ∧ d.year_range_end) {
      *year_range_end = *d.year_range_end;
    }
    else if (year_range_end ∧ ¬d.year_range_end) {
      delete year_range_end;
      year_range_end = 0;
    }
    else if (¬year_range_end ∧ d.year_range_end) {
      year_range_end = new short(*d.year_range_end);
    }
    if (year ∧ d.year) {
      *year = *d.year;
    }
    else if (year ∧ ¬d.year) {
```

```
    delete year;
    year = 0;
  }
  else if (¬year ∧ d.year) {
    year = new short(∗d.year);
  }
  if (month ∧ d.month) {
    ∗month = ∗d.month;
  }
  else if (month ∧ ¬d.month) {
    delete month;
    month = 0;
  }
  else if (¬month ∧ d.month) {
    month = new unsigned short(∗d.month);
  }
  if (day ∧ d.day) {
    ∗day = ∗d.day;
  }
  else if (day ∧ ¬d.day) {
    delete day;
    day = 0;
  }
  else if (¬day ∧ d.day) {
    day = new unsigned short(∗d.day);
  }
  if (hour ∧ d.hour) {
    ∗hour = ∗d.hour;
  }
  else if (hour ∧ ¬d.hour) {
    delete hour;
    hour = 0;
  }
  else if (¬hour ∧ d.hour) {
    hour = new unsigned short(∗d.hour);
  }
  if (minute ∧ d.minute) {
    ∗minute = ∗d.minute;
  }
  else if (minute ∧ ¬d.minute) {
    delete minute;
    minute = 0;
  }
  else if (¬minute ∧ d.minute) {
    minute = new unsigned short(∗d.minute);
  }
  if (second ∧ d.second) {
    ∗second = ∗d.second;
  }
  else if (second ∧ ¬d.second) {
    delete second;
    second = 0;
```

```
    }
    else if (¬second ∧ d.second) {
      second = new float(∗d.second);
    }
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "Exiting␣‘Date_Time_Type’␣assignment␣operator." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    temp_strm.str("");
#endif
    return this;
  }   /∗ End of the default Date_Time_Type assignment operator definition. ∗/
```

38. Output operator. [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this function.

⟨ Declare non-member functions for **Date_Time_Type** 38 ⟩ ≡
  ostream & **operator**≪(*ostream* & *o*, **Date_Time_Type** &*d*);

This code is used in sections 43 and 44.

39.

```
⟨ Define non-member functions for Date_Time_Type 39 ⟩ ≡
  ostream & operator<<(ostream & o, Date_Time_Type &d)
  {
#if 0     /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣‘Date_Time_Type’␣output␣operator." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    temp_strm.str("");
#endif
    o << "year_range_begin␣==␣";
    if (d.year_range_begin) o << *d.year_range_begin;
    else o << "NULL";
    o << endl;
    o << "year_range_end␣==␣";
    if (d.year_range_end) o << *d.year_range_end;
    else o << "NULL";
    o << endl;
    o << "year␣==␣";
    if (d.year) o << *d.year;
    else o << "NULL";
    o << endl;
    o << "month␣==␣";
    if (d.month) o << *d.month;
    else o << "NULL";
    o << endl;
    o << "day␣==␣";
    if (d.day) o << *d.day;
    else o << "NULL";
    o << endl;
    o << "hour␣==␣";
    if (d.hour) o << *d.hour;
    else o << "NULL";
    o << endl;
    o << "minute␣==␣";
    if (d.minute) o << *d.minute;
    else o << "NULL";
    o << endl;
    o << "second␣==␣";
    if (d.second) o << *d.second;
    else o << "NULL";
    o << endl;
    return o;
  }    /* End of operator<< definition. */
```

This code is used in section 43.

40. Show. [LDF 2006.12.15.]

Log

[2006.12.15.] Added this function.

⟨ Declare **Date_Time_Type** functions 32 ⟩ +≡
int *show*(**string** *s* = "`Date_Time_Type:'");

41.

⟨ Define **Date_Time_Type** functions 33 ⟩ +≡

```
int Date_Time_Type::show(string s)
{
  stringstream temp_strm;
  temp_strm ≪ s ≪ endl ≪ "‘year_range_begin’␣␣==␣";
  if (year_range_begin ≡ 0) temp_strm ≪ "NULL";
  else temp_strm ≪ *static_cast⟨short *⟩(year_range_begin);
  temp_strm ≪ endl;
  temp_strm ≪ "‘year_range_end’␣␣==␣";
  if (year_range_end ≡ 0) temp_strm ≪ "NULL";
  else temp_strm ≪ *static_cast⟨short *⟩(year_range_end);
  temp_strm ≪ endl;
  temp_strm ≪ "‘year’␣␣==␣";
  if (year ≡ 0) temp_strm ≪ "NULL";
  else temp_strm ≪ *static_cast⟨short *⟩(year);
  temp_strm ≪ endl;
  temp_strm ≪ "‘month’␣␣==␣";
  if (month ≡ 0) temp_strm ≪ "NULL";
  else temp_strm ≪ *static_cast⟨unsigned short *⟩(month);
  temp_strm ≪ endl;
  temp_strm ≪ "‘day’␣␣==␣";
  if (day ≡ 0) temp_strm ≪ "NULL";
  else temp_strm ≪ *static_cast⟨unsigned short *⟩(day);
  temp_strm ≪ endl;
  temp_strm ≪ "‘hour’␣␣==␣";
  if (hour ≡ 0) temp_strm ≪ "NULL";
  else temp_strm ≪ *static_cast⟨unsigned short *⟩(hour);
  temp_strm ≪ endl;
  temp_strm ≪ "‘minute’␣␣==␣";
  if (minute ≡ 0) temp_strm ≪ "NULL";
  else temp_strm ≪ *static_cast⟨unsigned short *⟩(minute);
  temp_strm ≪ endl;
  temp_strm ≪ "‘second’␣␣==␣";
  if (second ≡ 0) temp_strm ≪ "NULL";
  else temp_strm ≪ *static_cast⟨float *⟩(second);
  temp_strm ≪ endl;
  cerr ≪ temp_strm.str();
  temp_strm.str("");
  return 0;
}   /* End of the default Date_Time_Type::show definition. */
```

42. Putting Date_Time_Type together. [LDF 2006.10.31.]

43. This is what's compiled.

⟨ Include files 10 ⟩
⟨ `dttmtype.web` 25 ⟩
using namespace std;
⟨ Forward declarations 13 ⟩
⟨ Declare **class Date_Time_Type** 29 ⟩
⟨ Declare non-member functions for **Date_Time_Type** 38 ⟩
⟨ Define **Date_Time_Type** functions 33 ⟩
⟨ Define non-member functions for **Date_Time_Type** 39 ⟩

44. This is what's written to `dttmtype.h`.

```
⟨ dttmtype.hh 44 ⟩ ≡
  ⟨ Preprocessor macro calls 19 ⟩
  using namespace std;
  ⟨ Forward declarations 13 ⟩
  ⟨ Declare class Date_Time_Type 29 ⟩
  ⟨ Declare non-member functions for Date_Time_Type 38 ⟩
```

45. Query_Type (querytyp.web). [LDF 2006.10.17.]

Log

[LDF 2006.10.31.] Created this file.

```
⟨ querytyp.web 45 ⟩ ≡
  static char id_string[] = "$Id: querytyp.web,v 1.5 2007/01/02 12:20:58 lfinsto1 Exp $";
```

This code is cited in sections 6 and 8.

This code is used in section 118.

46. Include files.

```
⟨ Include files 10 ⟩ +≡
#include "localldf.h"
#ifdef NON_WIN_LDF
#include "nonwin.h"
#else
#include "stdafx.h"
#endif
#include "parser.hxx"
#include "dttmtype.h"
#include "idtype.h"
#include "scnrtype.h"
#include "scanner.h"
```

47. Preprocessor macro calls.

```
⟨ Preprocessor macro calls 19 ⟩ +≡
#ifdef WIN_LDF
#pragma once
#endif
```

48. Forward declarations. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this section.

```
⟨ Forward declarations 13 ⟩ +≡
  class Scanner_Type;
  typedef Scanner_Type *Scanner_Node;
  class Query_Type;
  typedef Query_Type *Query_Node;
  class Id_Type;
```

typedef Id_Type ∗**Id_Node**;

49. class Query_Type declaration. . [LDF 2006.10.17.]
The forward declaration of **Query_Type** is needed in order to be able to define **Query_Node** using **typedef**. [LDF 2006.10.31.]

Log

[LDF 2006.10.31.] Added **typedef Query_Type** ∗**Query_Node**.

[LDF 2006.10.31.] Working on this declaration.

[LDF 2006.11.13.] Removed **Query_Node** *next*. It was redundant, since *and_node* serves the same purpose.

[LDF 2006.11.13.] Added **vector**⟨**unsigned short**⟩ *target_types* and **static map**⟨**unsigned short**, **string**⟩ *target_type_map*.

[LDF 2006.11.13.] Added **unsigned short** *field_type* and **static map**⟨**unsigned short**, **string**⟩ *field_type_map*.

[LDF 2006.11.21.] Added **string** *name* and **Id_Node** *id_node*.

[LDF 2006.11.22.] Added **unsigned short** *query_ctr*.

[LDF 2006.12.12.] Changed **unsigned short** array *match_values* to the non-array **unsigned short** *match_value*.

[LDF 2006.12.12.] Added **static map**⟨**unsigned short**, **string**⟩ *match_value_map*.

⟨ Declare **class Query_Type** 49 ⟩ ≡
  class Query_Type;
  typedef Query_Type ∗**Query_Node**; **class Query_Type** { ⟨ **friend** declarations for **class Query_Type** 51 ⟩
private: **unsigned short** *query_type*;
  unsigned short *field_type*;
  unsigned short *value_type*;
  vector⟨**unsigned short**⟩ *target_types*;
  static map⟨**unsigned short**, **string**⟩ *query_type_map*;
  static map⟨**unsigned short**, **string**⟩ *field_type_map*;
  static map⟨**unsigned short**, **string**⟩ *value_type_map*;
  static map⟨**unsigned short**, **string**⟩ *target_type_map*;
  static map⟨**unsigned short**, **string**⟩ *match_value_map*;
  Id_Node *id_node*;
  bool *negated*;
  string *name*;
  void ∗*value*;
  Query_Node *up*;
  Query_Node *and_node*;
  Query_Node *or_node*;
  Query_Node *xor_node*;
  unsigned short *match_value*;

See also section 50.
This code is used in sections 118 and 119.

50.

⟨ Declare **class Query_Type** 49 ⟩ +≡
Scanner_Node *scanner_node*;
unsigned short *query_ctr*;
public: ⟨ Declare **static Query_Type** constants 53 ⟩
⟨ Declare **static Query_Type** variables 74 ⟩
⟨ Declare **Query_Type** functions 82 ⟩
} ;

51. friend declarations for class Query_Type. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

[LDF 2006.11.01.] Changed the name of **Scan_Parse** :: *start_local_query_func* to **Scan_Parse** :: *start_local_database_query_func* and the name of **Scan_Parse** :: *end_local_query_func* to **Scan_Parse** :: *end_query_func*.

[LDF 2006.11.01.] Added the **friend** declaration of **Scan_Parse** :: *declare_variable_func*.

[LDF 2006.11.01.] Added the **char** ∗*name* argument to **Scan_Parse** :: *declare_variable_func*.

[LDF 2006.11.02.] Changed the return value of **Scan_Parse** :: *declare_variable_func* from **int** to **Id_Node**.

[LDF 2006.11.02.] Added the **friend** declaration of **Scan_Parse** :: *variable_func*.

[LDF 2006.11.02.] Added the **friend** declaration of **Scan_Parse** :: *query_assignment_func_0*.

[LDF 2006.11.14.] Added the **friend** declaration of *yyparse*.

[LDF 2006.12.15.] Added the **friend** declaration of **Scan_Parse** :: *datetime_assignment_func_0*.

[LDF 2006.12.18.] Added the **friend** declaration of **Scan_Parse** :: *datetime_assignment_func_1*.

[LDF 2006.12.19.] Added the **friend** declaration of **Scan_Parse** :: *query_assignment_func_1*.

⟨ **friend** declarations for **class Query_Type** 51 ⟩ ≡
friend int *yyparse*(**void** ∗);
friend void ∗**Scan_Parse** :: *variable_func*(**void** ∗*v*, **char** ∗*name*);
friend Id_Node Scan_Parse :: *declare_variable_func*(**void** ∗*v*, **const unsigned short** *type*, **char** ∗*name*);
friend void ∗**Scan_Parse** :: *query_assignment_func_0*(**void** ∗*v*, **void** ∗*object*, **unsigned int** *assignment_type*, **unsigned int** *arg_0*, **unsigned int** *arg_1*, **void** ∗*value*, **bool** *negate*);
friend int Scan_Parse :: *query_assignment_func_1*(**Scanner_Node** *scanner_node*, **Id_Node** &*curr_id_node*, **void** ∗&*field_specifier*, **int** *assignment_operator*, **int** *negation_optional*, **int** *match_term_optional*, **void** ∗&*v*, **int** *type*);
friend int Scan_Parse :: *start_local_database_query_func*(**void** ∗*v*);
friend int Scan_Parse :: *end_query_func*(**void** ∗*v*);
friend int Scan_Parse :: *datetime_assignment_func_0*(**void** ∗*v*, **Date_Time_Node** &*curr_date_time_node*, **int** *specifier*, **int** *op*, **void** ∗*val*, **int** *type*);
friend int Scan_Parse :: *datetime_assignment_func_1*(**void** ∗*v*, **Date_Time_Node** &*d*, **int** *op*, **void** ∗&*vec*);

This code is used in section 49.

52. Declare static Query_Type constants and variables. [LDF 2006.10.31.]

53. Types for *value_type*, *field_type*, *query_type*, and *target_type*. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

⟨ Declare **static Query_Type** constants 53 ⟩ ≡
 static const unsigned short `QUERY_TYPE_NULL_TYPE`;

See also sections 55, 58, 60, 61, 62, 69, 71, and 78.

This code is used in section 50.

54.

⟨ Initialize **static Query_Type** constants 54 ⟩ ≡
 const unsigned short Query_Type :: `QUERY_TYPE_NULL_TYPE` = 0;

See also sections 56, 63, 64, 66, 67, 68, 70, 72, and 79.

This code is used in section 118.

55. Types for *query_type*. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

[LDF 2006.11.13.] Added **static const unsigned short** `TOP_TYPE`.

⟨ Declare **static Query_Type** constants 53 ⟩ +≡
 static const unsigned short `TOP_TYPE`;
 static const unsigned short `AND_TYPE`;
 static const unsigned short `OR_TYPE`;
 static const unsigned short `XOR_TYPE`;

56.

⟨ Initialize **static Query_Type** constants 54 ⟩ +≡
 const unsigned short Query_Type :: `TOP_TYPE` = 1;
 const unsigned short Query_Type :: `AND_TYPE` = 2;
 const unsigned short Query_Type :: `OR_TYPE` = 3;
 const unsigned short Query_Type :: `XOR_TYPE` = 4;

57. Types for *field_type*. [LDF 2006.11.13.]

Log

[LDF 2006.11.13.] Added this section with the declarations of `AUTHOR_FIELD`, `CONTRIBUTOR_FIELD`, `TITLE_FIELD`, and `SUBJECT_FIELD`.

[LDF 2006.12.05.] Added **static const unsigned short** `CREATOR_FIELD`.

[LDF 2006.12.07.] Added the **static const unsigned** *shorts* `AUTHOR_SURNAME_FIELD`, `AUTHOR_GIVEN_NAME_FIELD`, `AUTHOR_PREFIX_FIELD`, `CONTRIBUTOR_SURNAME_FIELD`, `CONTRIBUTOR_GIVEN_NAME_FIELD`, and `CONTRIBUTOR_PREFIX_FIELD`.

[LDF 2006.12.12.] Added the **static const unsigned short** `AUTHOR_SURNAME_FIELD`,

[LDF 2006.12.12.] Added the **static const unsigned short** `MAIN_CANONICAL_TITLE_FIELD`.

[LDF 2006.12.12.] Added **static const unsigned** *shorts* to refer to additional tables from the OAI database (dc_test).

[LDF 2006.12.12.] Added **static const unsigned** *shorts* to refer to additional tables from the PICA database (PICA_DB).

---

58. Tables. [LDF 2006.12.13.]

⟨ Declare **static Query_Type** constants 53 ⟩ +≡

```
static const unsigned short ACCESS_NUMBER_FIELD;
static const unsigned short AUTHOR_FIELD;
static const unsigned short BIBLIOGRAPHIC_TYPE_FIELD;
static const unsigned short CALL_NUMBER_FIELD;
static const unsigned short CLASSIFICATION_FIELD;
static const unsigned short COMPANY_FIELD;
static const unsigned short CONTENT_SUMMARY_FIELD;
static const unsigned short CONTRIBUTOR_FIELD;
static const unsigned short CREATOR_FIELD;
static const unsigned short DATABASE_PROVIDER_FIELD;
static const unsigned short DESCRIPTION_FIELD;
static const unsigned short EXEMPLAR_PRODUCTION_NUMBER_FIELD;
static const unsigned short IDENTIFIER_FIELD;
static const unsigned short INSTITUTION_FIELD;
static const unsigned short LANGUAGE_FIELD;
static const unsigned short MAIN_CANONICAL_TITLE_FIELD;
static const unsigned short PERMUTATION_PATTERN_FIELD;
static const unsigned short PHYSICAL_DESCRIPTION_FIELD;
static const unsigned short PERSON_FIELD;
static const unsigned short PUBLISHER_FIELD;
static const unsigned short RECORD_FIELD;
static const unsigned short REMOTE_ACCESS_FIELD;
static const unsigned short RIGHTS_FIELD;
static const unsigned short SOURCE_FIELD;
static const unsigned short SUBJECT_FIELD;
static const unsigned short SUPERORDINATE_ENTITIES_FIELD;
static const unsigned short TITLE_FIELD;
static const unsigned short TYPE_FIELD;
```

59. Columns. [LDF 2006.12.13.]

60. Authors. [LDF 2006.12.13.]

⟨ Declare **static Query_Type** constants 53 ⟩ +≡

```
static const unsigned short AUTHOR_SURNAME_FIELD;
static const unsigned short AUTHOR_GIVEN_NAME_FIELD;
static const unsigned short AUTHOR_PREFIX_FIELD;
```

61. Contributors. [LDF 2006.12.14.]

⟨ Declare **static Query_Type** constants 53 ⟩ +≡

```
static const unsigned short CONTRIBUTOR_SURNAME_FIELD;
static const unsigned short CONTRIBUTOR_GIVEN_NAME_FIELD;
static const unsigned short CONTRIBUTOR_PREFIX_FIELD;
```

62. Records. [LDF 2006.12.14.]

⟨ Declare **static Query_Type** constants 53 ⟩ +≡

```
static const unsigned short RECORD_ID_FIELD;
static const unsigned short RECORD_ELN_ORIGINAL_ENTRY_FIELD;
static const unsigned short RECORD_ELN_MOST_RECENT_CHANGE_FIELD;
static const unsigned short RECORD_ELN_STATUS_CHANGE_FIELD;
static const unsigned short RECORD_IDENTIFICATION_NUMBER_FIELD;
static const unsigned short RECORD_DATE_ORIGINAL_ENTRY_FIELD;
static const unsigned short RECORD_DATE_MOST_RECENT_CHANGE_FIELD;
static const unsigned short RECORD_DATE_STATUS_CHANGE_FIELD;
static const unsigned short RECORD_SOURCE_ID_FIELD;
static const unsigned short RECORD_YEAR_APPEARANCE_BEGIN_FIELD;
static const unsigned short RECORD_YEAR_APPEARANCE_END_FIELD;
static const unsigned short RECORD_YEAR_APPEARANCE_RAK_WB_FIELD;
static const unsigned short RECORD_YEAR_APPEARANCE_ORIGINAL_FIELD;
```

63. Initialization. [LDF 2006.12.13.]

⟨ Initialize **static Query_Type** constants 54 ⟩ +≡

64. Tables. [LDF 2006.12.13.]

⟨ Initialize **static Query_Type** constants 54 ⟩ +≡
const unsigned short Query_Type :: ACCESS_NUMBER_FIELD = 1;
const unsigned short Query_Type :: AUTHOR_FIELD = 2;
const unsigned short Query_Type :: BIBLIOGRAPHIC_TYPE_FIELD = 3;
const unsigned short Query_Type :: CALL_NUMBER_FIELD = 4;
const unsigned short Query_Type :: CLASSIFICATION_FIELD = 5;
const unsigned short Query_Type :: COMPANY_FIELD = 6;
const unsigned short Query_Type :: CONTENT_SUMMARY_FIELD = 7;
const unsigned short Query_Type :: CONTRIBUTOR_FIELD = 8;
const unsigned short Query_Type :: CREATOR_FIELD = 9;
const unsigned short Query_Type :: DATABASE_PROVIDER_FIELD = 10;
const unsigned short Query_Type :: DESCRIPTION_FIELD = 11;
const unsigned short Query_Type :: EXEMPLAR_PRODUCTION_NUMBER_FIELD = 12;
const unsigned short Query_Type :: IDENTIFIER_FIELD = 13;
const unsigned short Query_Type :: INSTITUTION_FIELD = 14;
const unsigned short Query_Type :: LANGUAGE_FIELD = 15;
const unsigned short Query_Type :: MAIN_CANONICAL_TITLE_FIELD = 16;
const unsigned short Query_Type :: PERMUTATION_PATTERN_FIELD = 17;
const unsigned short Query_Type :: PERSON_FIELD = 18;
const unsigned short Query_Type :: PHYSICAL_DESCRIPTION_FIELD = 19;
const unsigned short Query_Type :: PUBLISHER_FIELD = 20;
const unsigned short Query_Type :: RECORD_FIELD = 21;
const unsigned short Query_Type :: REMOTE_ACCESS_FIELD = 22;
const unsigned short Query_Type :: RIGHTS_FIELD = 23;
const unsigned short Query_Type :: SOURCE_FIELD = 24;
const unsigned short Query_Type :: SUPERORDINATE_ENTITIES_FIELD = 25;
const unsigned short Query_Type :: TITLE_FIELD = 26;
const unsigned short Query_Type :: SUBJECT_FIELD = 27;
const unsigned short Query_Type :: TYPE_FIELD = 28;

65. Columns. [LDF 2006.12.13.]

66. Authors. [LDF 2006.12.13.]

⟨ Initialize **static Query_Type** constants 54 ⟩ +≡
const unsigned short Query_Type :: AUTHOR_SURNAME_FIELD = 200;
const unsigned short Query_Type :: AUTHOR_GIVEN_NAME_FIELD = 201;
const unsigned short Query_Type :: AUTHOR_PREFIX_FIELD = 202;

67. Contributors. [LDF 2006.12.13.]

⟨ Initialize **static Query_Type** constants 54 ⟩ +≡
const unsigned short Query_Type :: CONTRIBUTOR_SURNAME_FIELD = 203;
const unsigned short Query_Type :: CONTRIBUTOR_GIVEN_NAME_FIELD = 204;
const unsigned short Query_Type :: CONTRIBUTOR_PREFIX_FIELD = 205;

68. Records. [LDF 2006.12.14.]

⟨ Initialize **static Query_Type** constants 54 ⟩ +≡
const unsigned short Query_Type :: RECORD_ID_FIELD = 500;
const unsigned short Query_Type :: RECORD_ELN_ORIGINAL_ENTRY_FIELD = 501;
const unsigned short Query_Type :: RECORD_ELN_MOST_RECENT_CHANGE_FIELD = 502;
const unsigned short Query_Type :: RECORD_ELN_STATUS_CHANGE_FIELD = 503;
const unsigned short Query_Type :: RECORD_IDENTIFICATION_NUMBER_FIELD = 504;
const unsigned short Query_Type :: RECORD_DATE_ORIGINAL_ENTRY_FIELD = 505;
const unsigned short Query_Type :: RECORD_DATE_MOST_RECENT_CHANGE_FIELD = 506;
const unsigned short Query_Type :: RECORD_DATE_STATUS_CHANGE_FIELD = 507;
const unsigned short Query_Type :: RECORD_SOURCE_ID_FIELD = 508;
const unsigned short Query_Type :: RECORD_YEAR_APPEARANCE_BEGIN_FIELD = 509;
const unsigned short Query_Type :: RECORD_YEAR_APPEARANCE_END_FIELD = 510;
const unsigned short Query_Type :: RECORD_YEAR_APPEARANCE_RAK_WB_FIELD = 501;
const unsigned short Query_Type :: RECORD_YEAR_APPEARANCE_ORIGINAL_FIELD = 512;

69. Types for *value_type*. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

⟨ Declare **static Query_Type** constants 53 ⟩ +≡
static const unsigned short INT_TYPE;
static const unsigned short FLOAT_TYPE;
static const unsigned short QUERY_TYPE_STRING_TYPE;
static const unsigned short DATE_TIME_TYPE;

70.

⟨ Initialize **static Query_Type** constants 54 ⟩ +≡
const unsigned short Query_Type :: INT_TYPE = 1;
const unsigned short Query_Type :: FLOAT_TYPE = 2;
const unsigned short Query_Type :: QUERY_TYPE_STRING_TYPE = 3;
const unsigned short Query_Type :: DATE_TIME_TYPE = 4;

71. Types for *target_types*. [LDF 2006.11.13.]

Log

[LDF 2006.11.13.] Added this section with the declarations of the **static const unsigned** *shorts* LOCAL_DATABASE_TARGET, LOCAL_SERVER_TARGET, REMOTE_DATABASE_TARGET, and REMOTE_SERVER_TARGET.

⟨ Declare **static Query_Type** constants 53 ⟩ +≡
static const unsigned short LOCAL_DATABASE_TARGET;
static const unsigned short LOCAL_SERVER_TARGET;
static const unsigned short REMOTE_DATABASE_TARGET;
static const unsigned short REMOTE_SERVER_TARGET;

72.

⟨ Initialize **static Query_Type** constants 54 ⟩ +≡
  const unsigned short Query_Type :: LOCAL_DATABASE_TARGET = 1;
  const unsigned short Query_Type :: LOCAL_SERVER_TARGET = 2;
  const unsigned short Query_Type :: REMOTE_DATABASE_TARGET = 3;
  const unsigned short Query_Type :: REMOTE_SERVER_TARGET = 4;

73. Flags. [LDF 2006.12.13.]

Log

[2006.12.13.] Added this section.

⟨ Preprocessor macro definitions 73 ⟩ ≡
#define QUERY_TYPE_BITSET_SIZE 96
This code is used in sections 118 and 119.

74.

⟨ Declare **static Query_Type** variables 74 ⟩ ≡
  static *bitset* < QUERY_TYPE_BITSET_SIZE > NULL_BITSET;
See also section 75.
This code is used in section 50.

75. Tables. [LDF 2006.12.13.]

⟨ Declare **static Query_Type** variables 74 ⟩ +≡

```
static bitset < QUERY_TYPE_BITSET_SIZE > ACCESS_NUMBER_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > AUTHOR_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > AUTHOR_SURNAME_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > AUTHOR_GIVEN_NAME_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > AUTHOR_PREFIX_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > BIBLIOGRAPHIC_TYPE_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > CALL_NUMBER_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > CLASSIFICATION_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > COMPANY_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > CONTENT_SUMMARY_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > CONTRIBUTOR_SURNAME_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > CONTRIBUTOR_GIVEN_NAME_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > CONTRIBUTOR_PREFIX_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > CREATOR_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > CONTRIBUTOR_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > DATABASE_PROVIDER_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > DESCRIPTION_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > EXEMPLAR_PRODUCTION_NUMBER_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > IDENTIFIER_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > INSTITUTION_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > LANGUAGE_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > MAIN_CANONICAL_TITLE_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > PERMUTATION_PATTERN_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > PERSON_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > PHYSICAL_DESCRIPTION_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > PUBLISHER_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_ID_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_ELN_ORIGINAL_ENTRY_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_ELN_MOST_RECENT_CHANGE_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_ELN_STATUS_CHANGE_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_IDENTIFICATION_NUMBER_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_DATE_ORIGINAL_ENTRY_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_DATE_MOST_RECENT_CHANGE_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_DATE_STATUS_CHANGE_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_SOURCE_ID_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_YEAR_APPEARANCE_BEGIN_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_YEAR_APPEARANCE_END_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_YEAR_APPEARANCE_RAK_WB_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RECORD_YEAR_APPEARANCE_ORIGINAL_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > REMOTE_ACCESS_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > RIGHTS_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > SOURCE_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > SUBJECT_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > SUPERORDINATE_ENTITIES_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > TITLE_FLAG;
static bitset < QUERY_TYPE_BITSET_SIZE > TYPE_FLAG;
```

76.

⟨ Initialize **static Query_Type** variables 76 ⟩ ≡

```
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::NULL_BITSET;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::ACCESS_NUMBER_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::AUTHOR_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::BIBLIOGRAPHIC_TYPE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::CALL_NUMBER_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::CLASSIFICATION_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::COMPANY_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::CONTENT_SUMMARY_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::CREATOR_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::CONTRIBUTOR_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::DATABASE_PROVIDER_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::DESCRIPTION_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::EXEMPLAR_PRODUCTION_NUMBER_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::IDENTIFIER_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::INSTITUTION_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::LANGUAGE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::MAIN_CANONICAL_TITLE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::PERMUTATION_PATTERN_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::PERSON_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::PHYSICAL_DESCRIPTION_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::PUBLISHER_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_ID_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_ELN_ORIGINAL_ENTRY_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_ELN_MOST_RECENT_CHANGE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_ELN_STATUS_CHANGE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_IDENTIFICATION_NUMBER_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_DATE_ORIGINAL_ENTRY_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_DATE_MOST_RECENT_CHANGE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_DATE_STATUS_CHANGE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_SOURCE_ID_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_YEAR_APPEARANCE_BEGIN_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_YEAR_APPEARANCE_END_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_YEAR_APPEARANCE_RAK_WB_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RECORD_YEAR_APPEARANCE_ORIGINAL_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::REMOTE_ACCESS_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::RIGHTS_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::SOURCE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::SUBJECT_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::SUPERORDINATE_ENTITIES_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::TITLE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::TYPE_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::AUTHOR_SURNAME_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::AUTHOR_GIVEN_NAME_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::AUTHOR_PREFIX_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::CONTRIBUTOR_SURNAME_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::CONTRIBUTOR_GIVEN_NAME_FLAG;
bitset < QUERY_TYPE_BITSET_SIZE > Query_Type::CONTRIBUTOR_PREFIX_FLAG;
```

See also section 77.

This code is used in section 118.

77. Initialize static Query_Type variables. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

[LDF 2006.11.13.] Added declaration of *target_type_map*.

[LDF 2006.11.13.] Added declaration of *field_type_map*.

[LDF 2006.12.12.] Added declaration of *match_value_map*.

⟨ Initialize **static Query_Type** variables 76 ⟩ +≡
  map⟨**unsigned short**, **string**⟩ **Query_Type** :: *query_type_map*;
  map⟨**unsigned short**, **string**⟩ **Query_Type** :: *field_type_map*;
  map⟨**unsigned short**, **string**⟩ **Query_Type** :: *value_type_map*;
  map⟨**unsigned short**, **string**⟩ **Query_Type** :: *target_type_map*;
  map⟨**unsigned short**, **string**⟩ **Query_Type** :: *match_value_map*;

78. Constants for *match_value*. [LDF 2006.12.11.]

Log

[LDF 2006.12.11.] Added this section with the declarations of the **static const unsigned** *shorts* `LIKE_VALUE` and `FREETEXT_VALUE`.

[LDF 2006.12.12.] Added **unsigned short** `CONTAINS_VALUE`.

⟨ Declare **static Query_Type** constants 53 ⟩ +≡
  static const unsigned short `CONTAINS_VALUE`;
  static const unsigned short `FREETEXT_VALUE`;
  static const unsigned short `LIKE_VALUE`;

79.

⟨ Initialize **static Query_Type** constants 54 ⟩ +≡
  const unsigned short Query_Type :: `CONTAINS_VALUE` = 1;
  const unsigned short Query_Type :: `FREETEXT_VALUE` = 2;
  const unsigned short Query_Type :: `LIKE_VALUE` = 3;

80. Query_Type functions. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

---

81. Constructor and setting functions. [LDF 2006.10.31.]

82. Default constructor. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this function.

⟨ Declare **Query_Type** functions 82 ⟩ ≡
  Query_Type(**void**);

See also sections 84, 86, 91, 95, 97, 101, 106, 110, and 111.

This code is used in section 50.

83.

⟨ Define **Query_Type** functions 83 ⟩ ≡

```
Query_Type::Query_Type(void)
{
    query_type = QUERY_TYPE_NULL_TYPE;
    field_type = QUERY_TYPE_NULL_TYPE;
    value_type = QUERY_TYPE_NULL_TYPE;
    negated = false;
    id_node = 0;
    value = 0;
    up = 0;
    and_node = 0;
    or_node = 0;
    xor_node = 0;
    query_ctr = 0;
    match_value = QUERY_TYPE_NULL_TYPE;
    scanner_node = 0;
    return;
}   /* End of default Query_Type constructor definition. */
```

See also sections 85, 87, 88, 89, 90, 92, 93, 94, 96, 98, 99, 100, 102, 103, 104, 107, 108, 109, 112, 113, 114, 115, 116, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, and 152.

This code is used in sections 118 and 154.

84. Set. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this function.

[LDF 2006.11.13.] Added **unsigned short** *ffield* and *ttarget* arguments. Made all arguments optional, except the first one, i.e., *qquery_type*.

⟨ Declare **Query_Type** functions 82 ⟩ +≡

```
int set(unsigned short qquery_type, unsigned short ffield_type = 0, unsigned short
    vvalue_type = 0, unsigned short ttarget_type = 0, bool nnegated = false, void
    *vvalue = 0, Query_Node uup = 0, Query_Node aand_node = 0, Query_Node
    oor_node = 0, Query_Node xxor_node = 0, Scanner_Node sscanner_node = 0);
```

85.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  int Query_Type::set(unsigned short qquery_type, unsigned short ffield_type, unsigned short
        vvalue_type, unsigned short target_type, bool nnegated, void *vvalue, Query_Node uup,
        Query_Node aand_node, Query_Node oor_node, Query_Node xxor_node, Scanner_Node
        sscanner_node)
  {
    match_value = QUERY_TYPE_NULL_TYPE;
    return 0;
  }   /* End of Query_Type::set definition. */
```

86. Destructor. [LDF 2006.10.31.]

To Do

[LDF 2006.12.12.] Change the way some *values* are handled, depending on the *field_type*. I've added dummy code for new fields that assumes that all of the associated values are pointers to **string**. This will probably not be the case.

Log

[2006.10.31.] Added this function.

[LDF 2006.11.14.] [LDF 2006.11.14.] !! BUG FIX: No longer writing debugging output to *scanner_node→log_strm*. This caused a "General Protection Fault" using GCC under Windows XP.

[LDF 2006.12.12.] Added code to account for additional **static const unsigned** *shorts* that refer to tables in the OAI database (dc_test).

[LDF 2006.12.12.] Added code to account for the tokens that refer to additional tables in the PICA database (PICA_DB).

```
⟨ Declare Query_Type functions 82 ⟩ +≡
  ~Query_Type(void);
```

87.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  Query_Type::~Query_Type(void){
#if 0   /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣'Query_Type'␣destructor." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    temp_strm.str("");
#endif
```

88. Please note that the **Query_Node** *up* isn't deleted! [LDF 2006.11.03.]

To Do

[LDF 2006.11.03.] Cast *value* to correct type and delete it.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  up = 0;
  if (value ≠ 0) {
    if (field_type ≡ ACCESS_NUMBER_FIELD ∨ field_type ≡ AUTHOR_FIELD ∨ field_type ≡
          BIBLIOGRAPHIC_TYPE_FIELD ∨ field_type ≡ CALL_NUMBER_FIELD ∨ field_type ≡
          CLASSIFICATION_FIELD ∨ field_type ≡ COMPANY_FIELD ∨ field_type ≡
          CONTENT_SUMMARY_FIELD ∨ field_type ≡ CONTRIBUTOR_FIELD ∨ field_type ≡
          CREATOR_FIELD ∨ field_type ≡ DATABASE_PROVIDER_FIELD ∨ field_type ≡
          DESCRIPTION_FIELD ∨ field_type ≡ EXEMPLAR_PRODUCTION_NUMBER_FIELD ∨ field_type ≡
          IDENTIFIER_FIELD∨field_type ≡ INSTITUTION_FIELD∨field_type ≡ LANGUAGE_FIELD∨field_type ≡
          MAIN_CANONICAL_TITLE_FIELD ∨ field_type ≡ PERMUTATION_PATTERN_FIELD ∨ field_type ≡
          PERSON_FIELD ∨ field_type ≡ PHYSICAL_DESCRIPTION_FIELD ∨ field_type ≡
          PUBLISHER_FIELD∨field_type ≡ RECORD_FIELD∨field_type ≡ REMOTE_ACCESS_FIELD∨field_type ≡
          RIGHTS_FIELD ∨ field_type ≡ SOURCE_FIELD ∨ field_type ≡ SUBJECT_FIELD ∨ field_type ≡
          SUPERORDINATE_ENTITIES_FIELD ∨ field_type ≡ TITLE_FIELD ∨ field_type ≡
          TYPE_FIELD ∨ field_type ≡ AUTHOR_SURNAME_FIELD ∨ field_type ≡
          AUTHOR_GIVEN_NAME_FIELD ∨ field_type ≡ AUTHOR_PREFIX_FIELD ∨ field_type ≡
          CONTRIBUTOR_SURNAME_FIELD ∨ field_type ≡ CONTRIBUTOR_GIVEN_NAME_FIELD ∨ field_type ≡
          CONTRIBUTOR_PREFIX_FIELD ∨ field_type ≡ RECORD_IDENTIFICATION_NUMBER_FIELD) {
      delete static_cast⟨string *⟩(value);
    }
    else if (field_type ≡ RECORD_ID_FIELD∨field_type ≡ RECORD_ELN_ORIGINAL_ENTRY_FIELD∨field_type ≡
          RECORD_ELN_MOST_RECENT_CHANGE_FIELD ∨ field_type ≡
          RECORD_ELN_STATUS_CHANGE_FIELD ∨ field_type ≡ RECORD_SOURCE_ID_FIELD ∨ field_type ≡
          RECORD_YEAR_APPEARANCE_BEGIN_FIELD ∨ field_type ≡ RECORD_YEAR_APPEARANCE_END_FIELD ∨
          field_type ≡ RECORD_YEAR_APPEARANCE_RAK_WB_FIELD ∨ field_type ≡
          RECORD_YEAR_APPEARANCE_ORIGINAL_FIELD) {
      delete static_cast⟨int *⟩(value);
    }
    else if (field_type ≡ RECORD_DATE_ORIGINAL_ENTRY_FIELD ∨ field_type ≡
          RECORD_DATE_MOST_RECENT_CHANGE_FIELD ∨ field_type ≡ RECORD_DATE_STATUS_CHANGE_FIELD)
      {
      delete static_cast⟨Date_Time_Node⟩(value);
    }
  }    /* if (value ≠ 0) */
  value = 0;
  delete and_node;
  and_node = 0;
  delete or_node;
  or_node = 0;
  delete xor_node;
  xor_node = 0;
```

89.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
#ifdef DEBUG_OUTPUT
  temp_strm << "Exiting␣‘Query_Type’␣destructor." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  temp_strm.str("");
#endif
```

90. DO NOT delete *scanner_node*! [LDF 2006.10.31.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
  scanner_node = 0;
  return; }     /* End of ~Query_Type definition. */
```

91. Assignment. [LDF 2006.11.15.]

Log

[2006.11.15.] Added this function.

⟨ Declare **Query_Type** functions 82 ⟩ +≡

```
  Query_Node operator=(const Query_Type &q);
```

92.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  Query_Node Query_Type::operator=(const Query_Type &q){
#if 0    /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
      stringstream temp_strm;
      scanner_node = q.scanner_node;
      query_ctr = q.query_ctr;
#ifdef DEBUG_OUTPUT
      temp_strm << "Entering␣‘Query_Type::operator=(const␣Query_Type)’." << endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      if (scanner_node) scanner_node→log_strm << temp_strm.str();
      temp_strm.str("");
#endif
      if (this ≡ &q) {
#ifdef DEBUG_OUTPUT
        temp_strm << "In␣‘Query_Type::operator=(const␣Query_Type)’:" << endl <<
            "Self-assignment.␣␣Returning␣‘this’." << endl;
        cerr_mutex.Lock();
        cerr << temp_strm.str();
        cerr_mutex.Unlock();
        if (scanner_node) scanner_node→log_strm << temp_strm.str();
        temp_strm.str("");
#endif
        return this;
      }
```

93.

Log

[LDF 2006.11.21.] Added this section. Now setting *name* to *q.name*.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
name = q.name;
if (value ≠ 0 ∧ value_type ≠ q.value_type) {
  if (value_type ≡ Query_Type::QUERY_TYPE_STRING_TYPE) {
    delete static_cast⟨string *⟩(value);
  }
  else if (value_type ≡ Query_Type::INT_TYPE) {
    delete static_cast⟨int *⟩(value);
  }
  else if (value_type ≡ Query_Type::FLOAT_TYPE) {
    delete static_cast⟨float *⟩(value);
  }
  else if (value_type ≡ Query_Type::DATE_TIME_TYPE) {
    delete static_cast⟨Date_Time_Node⟩(value);
  }
  value = 0;
}    /* if (value ≠ 0 ∧ value_type ≠ q.value_type) */
query_type = q.query_type;
field_type = q.field_type;
value_type = q.value_type;
negated = q.negated;
target_types.clear();
for (vector⟨unsigned short⟩::const_iterator iter = q.target_types.begin(); iter ≠ q.target_types.end();
       ++iter) target_types.push_back(*iter);
if (q.value ≠ 0 ∧ value_type ≡ QUERY_TYPE_STRING_TYPE) {
  if (value ≡ 0) value = static_cast⟨void *⟩(new string);
  *static_cast⟨string *⟩(value) = *static_cast⟨string *⟩(q.value);
}    /* if */
else if (q.value ≠ 0 ∧ value_type ≡ INT_TYPE) {
  if (value ≡ 0) value = static_cast⟨void *⟩(new int);
  *static_cast⟨int *⟩(value) = *static_cast⟨int *⟩(q.value);
}
else if (q.value ≠ 0 ∧ value_type ≡ FLOAT_TYPE) {
  if (value ≡ 0) value = static_cast⟨void *⟩(new float);
  *static_cast⟨float *⟩(value) = *static_cast⟨float *⟩(q.value);
}
else if (q.value ≠ 0 ∧ value_type ≡ DATE_TIME_TYPE) {
  if (value ≡ 0) value = static_cast⟨void *⟩(new Date_Time_Type);
  *static_cast⟨Date_Time_Node⟩(value) = *static_cast⟨Date_Time_Node⟩(q.value);
}
else {
  value_type = QUERY_TYPE_NULL_TYPE;
  value = 0;
}
scanner_node = q.scanner_node;
delete and_node;
```

```
and_node = 0;
delete or_node;
or_node = 0;
delete xor_node;
xor_node = 0;
if (q.and_node) {
  and_node = new Query_Type;
  *and_node = *q.and_node;
  and_node→up = this;
}
if (q.or_node) {
  or_node = new Query_Type;
  *or_node = *q.or_node;
  or_node→up = this;
}
if (q.xor_node) {
  xor_node = new Query_Type;
  *xor_node = *q.xor_node;
  xor_node→up = this;
}
#ifdef DEBUG_OUTPUT
temp_strm ≪ "Exiting␣‘Query_Type::operator=(const␣Query_Type)’." ≪ endl;
cerr_mutex.Lock();
cerr ≪ temp_strm.str();
cerr_mutex.Unlock();
if (scanner_node) scanner_node→log_strm ≪ temp_strm.str();
temp_strm.str("");
#endif
```

94.

⟨ Define **Query_Type** functions 83 ⟩ +≡
  match_value = *q*.*match_value*;
  return *this*; }   /* End of **Query_Type** :: **operator**=(**const Query_Type** *q*) definition. */

95. Initialize type maps. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this function.

[LDF 2006.11.13.] Added code for initializing *target_type_map*.

[LDF 2006.11.13.] Added code for initializing *field_type_map*.

[LDF 2006.12.12.] Added code for `MAIN_CANONICAL_TITLE_FIELD`.

[LDF 2006.12.12.] Added code for initializing *match_value_map*.

[LDF 2006.12.12.] Added code for additional **static const unsigned** *shorts* that refer to tables in the OAI database (dc_test).

[LDF 2006.12.12.] Added code to account for for additional **static const unsigned** *shorts* that refer to tables in the PICA database (PICA_DB).

⟨ Declare **Query_Type** functions 82 ⟩ +≡
  static int *initialize_type_maps*(**void**);

96.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
int Query_Type::initialize_type_maps(void)
{
  query_type_map[QUERY_TYPE_NULL_TYPE] = "QUERY_TYPE_NULL_TYPE";
  query_type_map[TOP_TYPE] = "TOP_TYPE";
  query_type_map[AND_TYPE] = "AND_TYPE";
  query_type_map[OR_TYPE] = "OR_TYPE";
  query_type_map[XOR_TYPE] = "XOR_TYPE";
  field_type_map[QUERY_TYPE_NULL_TYPE] = "QUERY_TYPE_NULL_TYPE";
  field_type_map[ACCESS_NUMBER_FIELD] = "ACCESS_NUMBER_FIELD";
  field_type_map[AUTHOR_FIELD] = "AUTHOR_FIELD";
  field_type_map[AUTHOR_SURNAME_FIELD] = "AUTHOR_SURNAME_FIELD";
  field_type_map[AUTHOR_GIVEN_NAME_FIELD] = "AUTHOR_GIVEN_NAME_FIELD";
  field_type_map[AUTHOR_PREFIX_FIELD] = "AUTHOR_PREFIX_FIELD";
  field_type_map[BIBLIOGRAPHIC_TYPE_FIELD] = "BIBLIOGRAPHIC_TYPE_FIELD";
  field_type_map[CALL_NUMBER_FIELD] = "CALL_NUMBER_FIELD";
  field_type_map[CLASSIFICATION_FIELD] = "CLASSIFICATION_FIELD";
  field_type_map[COMPANY_FIELD] = "COMPANY_FIELD";
  field_type_map[CONTENT_SUMMARY_FIELD] = "CONTENT_SUMMARY_FIELD";
  field_type_map[CONTRIBUTOR_FIELD] = "CONTRIBUTOR_FIELD";
  field_type_map[CONTRIBUTOR_SURNAME_FIELD] = "CONTRIBUTOR_SURNAME_FIELD";
  field_type_map[CONTRIBUTOR_GIVEN_NAME_FIELD] = "CONTRIBUTOR_GIVEN_NAME_FIELD";
  field_type_map[CONTRIBUTOR_PREFIX_FIELD] = "CONTRIBUTOR_PREFIX_FIELD";
  field_type_map[CREATOR_FIELD] = "CREATOR_FIELD";
  field_type_map[DATABASE_PROVIDER_FIELD] = "DATABASE_PROVIDER_FIELD";
  field_type_map[DESCRIPTION_FIELD] = "DESCRIPTION_FIELD";
  field_type_map[EXEMPLAR_PRODUCTION_NUMBER_FIELD] = "EXEMPLAR_PRODUCTION_NUMBER_FIELD";
  field_type_map[IDENTIFIER_FIELD] = "IDENTIFIER_FIELD";
  field_type_map[INSTITUTION_FIELD] = "INSTITUTION_FIELD";
  field_type_map[LANGUAGE_FIELD] = "LANGUAGE_FIELD";
  field_type_map[MAIN_CANONICAL_TITLE_FIELD] = "MAIN_CANONICAL_TITLE_FIELD";
  field_type_map[PERMUTATION_PATTERN_FIELD] = "PERMUTATION_PATTERN_FIELD";
  field_type_map[PERSON_FIELD] = "PERSON_FIELD";
  field_type_map[PHYSICAL_DESCRIPTION_FIELD] = "PHYSICAL_DESCRIPTION_FIELD";
  field_type_map[PUBLISHER_FIELD] = "PUBLISHER_FIELD";
  field_type_map[RECORD_FIELD] = "RECORD_FIELD";
  field_type_map[RECORD_IDENTIFICATION_NUMBER_FIELD] = "RECORD_IDENTIFICATION_NUMBER_FIELD";
  field_type_map[RECORD_DATE_MOST_RECENT_CHANGE_FIELD] =
      "RECORD_DATE_MOST_RECENT_CHANGE_FIELD";
  field_type_map[RECORD_DATE_ORIGINAL_ENTRY_FIELD] = "RECORD_DATE_ORIGINAL_ENTRY_FIELD";
  field_type_map[RECORD_DATE_STATUS_CHANGE_FIELD] = "RECORD_DATE_STATUS_CHANGE_FIELD";
  field_type_map[RECORD_ELN_STATUS_CHANGE_FIELD] = "RECORD_ELN_STATUS_CHANGE_FIELD";
  field_type_map[REMOTE_ACCESS_FIELD] = "REMOTE_ACCESS_FIELD";
  field_type_map[RIGHTS_FIELD] = "RIGHTS_FIELD";
  field_type_map[SOURCE_FIELD] = "SOURCE_FIELD";
  field_type_map[SUBJECT_FIELD] = "SUBJECT_FIELD";
  field_type_map[SUPERORDINATE_ENTITIES_FIELD] = "SUPERORDINATE_ENTITIES_FIELD";
  field_type_map[TITLE_FIELD] = "TITLE_FIELD";
  field_type_map[TYPE_FIELD] = "TYPE_FIELD";
  value_type_map[QUERY_TYPE_NULL_TYPE] = "QUERY_TYPE_NULL_TYPE";
  value_type_map[INT_TYPE] = "INT_TYPE";
```

value_type_map[FLOAT_TYPE] = "FLOAT_TYPE";
value_type_map[QUERY_TYPE_STRING_TYPE] = "QUERY_TYPE_STRING_TYPE";
value_type_map[DATE_TIME_TYPE] = "DATE_TIME_TYPE";
target_type_map[LOCAL_DATABASE_TARGET] = "LOCAL_DATABASE_TARGET";
target_type_map[LOCAL_SERVER_TARGET] = "LOCAL_SERVER_TARGET";
target_type_map[REMOTE_DATABASE_TARGET] = "REMOTE_DATABASE_TARGET";
target_type_map[REMOTE_SERVER_TARGET] = "REMOTE_SERVER_TARGET";
match_value_map[QUERY_TYPE_NULL_TYPE] = "QUERY_TYPE_NULL_TYPE";
match_value_map[CONTAINS_VALUE] = "CONTAINS_VALUE";
match_value_map[FREETEXT_VALUE] = "FREETEXT_VALUE";
match_value_map[LIKE_VALUE] = "LIKE_VALUE";
return 0;
} /* End of **Query_Type** :: *initialize_type_maps* definition. */

97. Initialize flags. [LDF 2006.12.13.]

Log

[2006.12.13.] Added this function.

⟨ Declare **Query_Type** functions 82 ⟩ +≡
static int *initialize_flags*(**void**);

98.

⟨ Define **Query_Type** functions 83 ⟩ +≡
int Query_Type :: *initialize_flags*(**void**){ **int** *shift_value* = 0;

99.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
ACCESS_NUMBER_FLAG = 1;
ACCESS_NUMBER_FLAG <<= shift_value++;
AUTHOR_FLAG = 1;
AUTHOR_FLAG <<= shift_value++;
AUTHOR_SURNAME_FLAG = 1;
AUTHOR_SURNAME_FLAG <<= shift_value++;
AUTHOR_GIVEN_NAME_FLAG = 1;
AUTHOR_GIVEN_NAME_FLAG <<= shift_value++;
AUTHOR_PREFIX_FLAG = 1;
AUTHOR_PREFIX_FLAG <<= shift_value++;
BIBLIOGRAPHIC_TYPE_FLAG = 1;
BIBLIOGRAPHIC_TYPE_FLAG <<= shift_value++;
CALL_NUMBER_FLAG = 1;
CALL_NUMBER_FLAG <<= shift_value++;
CLASSIFICATION_FLAG = 1;
CLASSIFICATION_FLAG <<= shift_value++;
COMPANY_FLAG = 1;
COMPANY_FLAG <<= shift_value++;
CONTENT_SUMMARY_FLAG = 1;
CONTENT_SUMMARY_FLAG <<= shift_value++;
CONTRIBUTOR_FLAG = 1;
CONTRIBUTOR_FLAG <<= shift_value++;
CONTRIBUTOR_SURNAME_FLAG = 1;
CONTRIBUTOR_SURNAME_FLAG <<= shift_value++;
CONTRIBUTOR_GIVEN_NAME_FLAG = 1;
CONTRIBUTOR_GIVEN_NAME_FLAG <<= shift_value++;
CONTRIBUTOR_PREFIX_FLAG = 1;
CONTRIBUTOR_PREFIX_FLAG <<= shift_value++;
CREATOR_FLAG = 1;
CREATOR_FLAG <<= shift_value++;
DATABASE_PROVIDER_FLAG = 1;
DATABASE_PROVIDER_FLAG <<= shift_value++;
DESCRIPTION_FLAG = 1;
DESCRIPTION_FLAG <<= shift_value++;
EXEMPLAR_PRODUCTION_NUMBER_FLAG = 1;
EXEMPLAR_PRODUCTION_NUMBER_FLAG <<= shift_value++;
IDENTIFIER_FLAG = 1;
IDENTIFIER_FLAG <<= shift_value++;
INSTITUTION_FLAG = 1;
INSTITUTION_FLAG <<= shift_value++;
LANGUAGE_FLAG = 1;
LANGUAGE_FLAG <<= shift_value++;
MAIN_CANONICAL_TITLE_FLAG = 1;
MAIN_CANONICAL_TITLE_FLAG <<= shift_value++;
PERMUTATION_PATTERN_FLAG = 1;
PERMUTATION_PATTERN_FLAG <<= shift_value++;
PERSON_FLAG = 1;
PERSON_FLAG <<= shift_value++;
PHYSICAL_DESCRIPTION_FLAG = 1;
PHYSICAL_DESCRIPTION_FLAG <<= shift_value++;
```

```
PUBLISHER_FLAG = 1;
PUBLISHER_FLAG <<= shift_value++;
RECORD_FLAG = 1;
RECORD_FLAG <<= shift_value++;
RECORD_ID_FLAG = 1;
RECORD_ID_FLAG <<= shift_value++;
RECORD_ELN_ORIGINAL_ENTRY_FLAG = 1;
RECORD_ELN_ORIGINAL_ENTRY_FLAG <<= shift_value++;
RECORD_ELN_MOST_RECENT_CHANGE_FLAG = 1;
RECORD_ELN_MOST_RECENT_CHANGE_FLAG <<= shift_value++;
RECORD_ELN_STATUS_CHANGE_FLAG = 1;
RECORD_ELN_STATUS_CHANGE_FLAG <<= shift_value++;
RECORD_IDENTIFICATION_NUMBER_FLAG = 1;
RECORD_IDENTIFICATION_NUMBER_FLAG <<= shift_value++;
RECORD_DATE_ORIGINAL_ENTRY_FLAG = 1;
RECORD_DATE_ORIGINAL_ENTRY_FLAG <<= shift_value++;
RECORD_DATE_MOST_RECENT_CHANGE_FLAG = 1;
RECORD_DATE_MOST_RECENT_CHANGE_FLAG <<= shift_value++;
RECORD_DATE_STATUS_CHANGE_FLAG = 1;
RECORD_DATE_STATUS_CHANGE_FLAG <<= shift_value++;
RECORD_SOURCE_ID_FLAG = 1;
RECORD_SOURCE_ID_FLAG <<= shift_value++;
RECORD_YEAR_APPEARANCE_BEGIN_FLAG = 1;
RECORD_YEAR_APPEARANCE_BEGIN_FLAG <<= shift_value++;
RECORD_YEAR_APPEARANCE_END_FLAG = 1;
RECORD_YEAR_APPEARANCE_END_FLAG <<= shift_value++;
RECORD_YEAR_APPEARANCE_RAK_WB_FLAG = 1;
RECORD_YEAR_APPEARANCE_RAK_WB_FLAG <<= shift_value++;
RECORD_YEAR_APPEARANCE_ORIGINAL_FLAG = 1;
RECORD_YEAR_APPEARANCE_ORIGINAL_FLAG <<= shift_value++;
REMOTE_ACCESS_FLAG = 1;
REMOTE_ACCESS_FLAG <<= shift_value++;
RIGHTS_FLAG = 1;
RIGHTS_FLAG <<= shift_value++;
SOURCE_FLAG = 1;
SOURCE_FLAG <<= shift_value++;
SUBJECT_FLAG = 1;
SUBJECT_FLAG <<= shift_value++;
SUPERORDINATE_ENTITIES_FLAG = 1;
SUPERORDINATE_ENTITIES_FLAG <<= shift_value++;
TITLE_FLAG = 1;
TITLE_FLAG <<= shift_value++;
TYPE_FLAG = 1;
TYPE_FLAG <<= shift_value++;
```

100. Error handling. `QUERY_TYPE_BITSET_SIZE` too small. [LDF 2006.12.13.]

```
⟨ Define Query_Type functions 83 ⟩ +≡
  if (shift_value > QUERY_TYPE_BITSET_SIZE) {
    cerr ≪ "ERROR!␣␣In␣‘Query_Type::initialize_flags’:" ≪ endl ≪
        "‘shift_value’␣>␣‘QUERY_TYPE_BITSET_SIZE’." ≪ endl ≪
        "Increase␣the␣size␣of␣‘QUERY_TYPE_BITSET_SIZE’." ≪ endl ≪
        "Exiting␣function␣with␣return␣value␣1." ≪ endl ≪ "Type␣<RETURN>␣to␣continue:␣";
    getchar( );
    return 1;
  }    /* if (shift_value > QUERY_TYPE_BITSET_SIZE) */
  return 0; }    /* Query_Type::initialize_flags */
```

101. Set field specifier. [LDF 2006.11.15.]

Log

[2006.11.15.] Added this function.

[LDF 2006.12.12.] Added code for `MAIN_CANONICAL_TITLE_FIELD`.

[LDF 2006.12.12.] Added code for more **static const unsigned** *shorts* that refer to tables in the OAI database (dc_test).

[LDF 2006.12.12.] Added code for more **static const unsigned** *shorts* that refer to tables in the PICA database (PICA_DB).

[LDF 2006.12.13.] Changed the name of this function from *set_field_type* to *set_field_specifier*.

```
⟨ Declare Query_Type functions 82 ⟩ +≡
  int set_field_specifier(void *v, bool traverse = false, Scanner_Node sscanner_node = 0);
```

102.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  int Query_Type::set_field_specifier(void *v, bool traverse, Scanner_Node sscanner_node){
#if 0    /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
      stringstream temp_strm; deque < int > *field_specifier = static_cast ⟨ deque < int >*> (v);
      unsigned int table = (*field_specifier)[0];
      unsigned int column;
      if (table ≡ ACCESS_NUMBER) field_type = ACCESS_NUMBER_FIELD;
      else if (table ≡ AUTHOR) {
        if (field_specifier→size( ) ≡ 1) field_type = AUTHOR_FIELD;
        else    /* field_specifier→size( ) ≠ 1 */
        {
          column = (*field_specifier)[1];
          if (column ≡ SURNAME) field_type = AUTHOR_SURNAME_FIELD;
          else if (column ≡ GIVEN_NAME) field_type = AUTHOR_GIVEN_NAME_FIELD;
          else if (column ≡ PREFIX) field_type = AUTHOR_PREFIX_FIELD;
        }    /* else (field_specifier→size( ) ≠ 1) */
      }    /* else if (table ≡ AUTHOR) */
      else if (table ≡ BIBLIOGRAPHIC_TYPE) field_type = BIBLIOGRAPHIC_TYPE_FIELD;
      else if (table ≡ CALL_NUMBER) field_type = CALL_NUMBER_FIELD;
      else if (table ≡ CLASSIFICATION) field_type = CLASSIFICATION_FIELD;
      else if (table ≡ COMPANY) field_type = COMPANY_FIELD;
      else if (table ≡ CONTENT_SUMMARY) field_type = CONTENT_SUMMARY_FIELD;
      else if (table ≡ CONTRIBUTOR) {
        if (field_specifier→size( ) ≡ 1) field_type = CONTRIBUTOR_FIELD;
        else    /* (field_specifier→size( ) ≠ 1) */
        {
          column = (*field_specifier)[1];
          if (column ≡ SURNAME) field_type = CONTRIBUTOR_SURNAME_FIELD;
          else if (column ≡ GIVEN_NAME) field_type = CONTRIBUTOR_GIVEN_NAME_FIELD;
          else if (column ≡ PREFIX) field_type = CONTRIBUTOR_PREFIX_FIELD;
        }    /* else ((field_specifier→size( ) ≠ 1)) */
      }    /* else if (table ≡ CONTRIBUTOR) */
      else if (table ≡ CREATOR) field_type = CREATOR_FIELD;
      else if (table ≡ DATABASE_PROVIDER) field_type = DATABASE_PROVIDER_FIELD;
      else if (table ≡ DESCRIPTION) field_type = DESCRIPTION_FIELD;
      else if (table ≡ EXEMPLAR_PRODUCTION_NUMBER)
        field_type = EXEMPLAR_PRODUCTION_NUMBER_FIELD;
      else if (table ≡ IDENTIFIER) field_type = IDENTIFIER_FIELD;
      else if (table ≡ INSTITUTION) field_type = INSTITUTION_FIELD;
      else if (table ≡ LANGUAGE) field_type = LANGUAGE_FIELD;
      else if (table ≡ MAIN_CANONICAL_TITLE) field_type = MAIN_CANONICAL_TITLE_FIELD;
      else if (table ≡ PERMUTATION_PATTERN) field_type = PERMUTATION_PATTERN_FIELD;
      else if (table ≡ PERSON) field_type = PERSON_FIELD;
      else if (table ≡ PHYSICAL_DESCRIPTION) field_type = PHYSICAL_DESCRIPTION_FIELD;
      else if (table ≡ PUBLISHER) field_type = PUBLISHER_FIELD;
```

103. *table* ≡ RECORD.

Log

[2006.12.14.] Added this section.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
else
  if (table ≡ RECORD) {
    if (field_specifier→size() ≡ 1) field_type = RECORD_FIELD;
    else    /* (field_specifier→size() ≠ 1) */
    {
      column = (*field_specifier)[1];
      if (column ≡ ID) field_type = RECORD_ID_FIELD;
      else if (column ≡ ELN_ORIGINAL_ENTRY) field_type = RECORD_ELN_ORIGINAL_ENTRY_FIELD;
      else if (column ≡ ELN_MOST_RECENT_CHANGE)
        field_type = RECORD_ELN_MOST_RECENT_CHANGE_FIELD;
      else if (column ≡ ELN_STATUS_CHANGE) field_type = RECORD_ELN_STATUS_CHANGE_FIELD;
      else if (column ≡ IDENTIFICATION_NUMBER)
        field_type = RECORD_IDENTIFICATION_NUMBER_FIELD;
      else if (column ≡ DATE_ORIGINAL_ENTRY) field_type = RECORD_DATE_ORIGINAL_ENTRY_FIELD;
      else if (column ≡ DATE_MOST_RECENT_CHANGE)
        field_type = RECORD_DATE_MOST_RECENT_CHANGE_FIELD;
      else if (column ≡ DATE_STATUS_CHANGE) field_type = RECORD_DATE_STATUS_CHANGE_FIELD;
      else if (column ≡ SOURCE_ID) field_type = RECORD_SOURCE_ID_FIELD;
      else if (column ≡ YEAR_APPEARANCE_BEGIN)
        field_type = RECORD_YEAR_APPEARANCE_BEGIN_FIELD;
      else if (column ≡ YEAR_APPEARANCE_END) field_type = RECORD_YEAR_APPEARANCE_END_FIELD;
      else if (column ≡ YEAR_APPEARANCE_RAK_WB)
        field_type = RECORD_YEAR_APPEARANCE_RAK_WB_FIELD;
      else if (column ≡ YEAR_APPEARANCE_ORIGINAL)
        field_type = RECORD_YEAR_APPEARANCE_ORIGINAL_FIELD;
    }    /* else ((field_specifier→size() ≠ 1)) */
  }    /* else if (table ≡ RECORD) */
```

104.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
else
  if (table ≡ REMOTE_ACCESS) field_type = REMOTE_ACCESS_FIELD;
  else if (table ≡ RIGHTS) field_type = RIGHTS_FIELD;
  else if (table ≡ SOURCE) field_type = SOURCE_FIELD;
  else if (table ≡ SUBJECT) field_type = SUBJECT_FIELD;
  else if (table ≡ SUPERORDINATE_ENTITIES) field_type = SUPERORDINATE_ENTITIES_FIELD;
  else if (table ≡ TITLE) field_type = TITLE_FIELD;
  else if (table ≡ TYPE) field_type = TYPE_FIELD;
  else if (table ≡ CONTRIBUTOR_SURNAME) field_type = CONTRIBUTOR_SURNAME_FIELD;
  else if (table ≡ CONTRIBUTOR_GIVEN_NAME) field_type = CONTRIBUTOR_GIVEN_NAME_FIELD;
  else if (table ≡ CONTRIBUTOR_PREFIX) field_type = CONTRIBUTOR_PREFIX_FIELD;
  else field_type = QUERY_TYPE_NULL_TYPE;
if (traverse) {
  if (and_node) and_node→set_field_specifier(v, true, scanner_node);
  if (or_node) or_node→set_field_specifier(v, true, scanner_node);
  if (xor_node) xor_node→set_field_specifier(v, true, scanner_node);
}
return 0;
#undef DEBUG_OUTPUT
}    /* End of Query_Type :: set_field_specifier definition. */
```

105. Functions for generating strings. [LDF 2006.11.21.]

Log

[LDF 2006.11.21.] Added this section.

---

106. Generate TEX string. [LDF 2006.11.21.]

Log

[2006.11.21.] Added this function.

⟨ Declare **Query_Type** functions 82 ⟩ +≡

```
string generate_tex_string(Scanner_Node scanner_node, stringstream *tex_strm);
```

107.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
string Query_Type::generate_tex_string(Scanner_Node scanner_node, stringstream *tex_strm){
#if 0    /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣‘Query_Type::generate_tex_string’." << endl <<
        "‘query_ctr’␣==␣" << query_ctr << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (scanner_node ≠ 0) scanner_node→log_strm << temp_strm.str();
    temp_strm.str("");
#endif
```

108.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
if (query_ctr ≡ 0) *tex_strm ≪ "\\setbox\\namebox=\\hbox{" ≪ name ≪ "}" ≪ endl;
*tex_strm ≪ "\\setbox\\querytypebox=\\hbox{" ≪ query_type_map[query_type] ≪ "}" ≪ endl;
*tex_strm ≪ "\\setbox\\fieldtypebox=\\hbox{" ≪ field_type_map[field_type] ≪ "}" ≪ endl;
*tex_strm ≪ "\\setbox\\valuetypebox=\\hbox{" ≪ value_type_map[value_type] ≪ "}" ≪ endl;
*tex_strm ≪ "\\setbox\\targettypesbox=\\hbox{";
if (target_types.size() ≤ 0) *tex_strm ≪ "QUERY_TYPE_NULL_TYPE";
else if (target_types.size() ≡ 1) {
  *tex_strm ≪ target_type_map[target_types[0]];
}
else {
  *tex_strm ≪ "\\vbox{";
  int i = 0;
  for (vector⟨unsigned short⟩::iterator iter = target_types.begin(); iter ≠ target_types.end();
        ++iter) {
    if (i++ > 0) *tex_strm ≪ "%" ≪ endl;
    *tex_strm ≪ "\\hbox{" ≪ target_type_map[*iter] ≪ "}";
  }    /* for */
  *tex_strm ≪ "}";
}    /* else */
*tex_strm ≪ "}" ≪ endl;
if (value_type ≡ QUERY_TYPE_STRING_TYPE) {
  *tex_strm ≪ "\\setbox\\valuebox=\\hbox{" ≪ *static_cast⟨string *⟩(value) ≪ "}" ≪ endl;
}
else if (value_type ≡ INT_TYPE) {
  *tex_strm ≪ "\\setbox\\valuebox=\\hbox{" ≪ *static_cast⟨int *⟩(value) ≪ "}" ≪ endl;
}
else if (value_type ≡ FLOAT_TYPE) {
  *tex_strm ≪ "\\setbox\\valuebox=\\hbox{" ≪ *static_cast⟨float *⟩(value) ≪ "}" ≪ endl;
}
else if (value_type ≡ DATE_TIME_TYPE) { *tex_strm ≪ "\\setbox\\valuebox=\\hbox{" ≪
    "Date\\_Time" # if 0    /* 1 */
*static_cast⟨Date_Time_Node⟩(value) # endif ≪ "}" ≪ endl; } *tex_strm ≪ "\\upctr=";
if (up) *tex_strm ≪ up→query_ctr;
else *tex_strm ≪ "0";
*tex_strm ≪ endl;
*tex_strm ≪ "\\andctr=";
if (and_node) *tex_strm ≪ and_node→query_ctr;
else *tex_strm ≪ "0";
*tex_strm ≪ endl;
*tex_strm ≪ "\\orctr=";
if (or_node) *tex_strm ≪ or_node→query_ctr;
else *tex_strm ≪ "0";
*tex_strm ≪ endl;
*tex_strm ≪ "\\xorctr=";
if (or_node) *tex_strm ≪ xor_node→query_ctr;
else *tex_strm ≪ "0";
*tex_strm ≪ endl;
*tex_strm ≪ "%" ≪ endl ≪ "\\chart{" ≪ query_ctr ≪ "}{" ≪ negated ≪ "}" ≪ endl ≪ "%" ≪ endl;
if (and_node) {
```

```
    *tex_strm ≪ "\\vskip\\queryskip\n%\n";
    *tex_strm ≪ "\\vskip\\ht\\targettypesbox\n" ≪ "\\vskip\\dp\\targettypesbox\n" ≪ "%\n";
    and_node→generate_tex_string(scanner_node, tex_strm);
  }
  if (or_node) {
    *tex_strm ≪ "\\vskip\\queryskip\n%\n";
    *tex_strm ≪ "\\vskip\\ht\\targettypesbox\n" ≪ "\\vskip\\dp\\targettypesbox\n" ≪ "%\n";
    or_node→generate_tex_string(scanner_node, tex_strm);
  }
  if (xor_node) {
    *tex_strm ≪ "\\vskip\\queryskip\n%\n";
    *tex_strm ≪ "\\vskip\\ht\\targettypesbox\n" ≪ "\\vskip\\dp\\targettypesbox\n" ≪ "%\n";
    xor_node→generate_tex_string(scanner_node, tex_strm);
  }
```

109. Return string. [LDF 2006.11.21.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "Exiting␣‘Query_Type::generate_tex_string’." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
  return tex_strm→str(); }     /* Query_Type::generate_tex_string */
```

110. Generate SQL string. [LDF 2006.12.01.]

Log

[LDF 2006.12.01.] Added this function.

[LDF 2006.12.12.] Added code for `MAIN_CANONICAL_TITLE_FLAG`.

[LDF 2006.12.12.] Added code to account for **static const unsigned** *shorts* that refer to additional tables in the OAI database (dc_test).

[LDF 2006.12.14.] Moved the definition of this function from this file (`querytyp.web`) to `qtgensql.web`.

⟨ Declare **Query_Type** functions 82 ⟩ +≡
int *generate_sql_string*(**Scanner_Node** *scanner_node*, **int** *database_type*, **stringstream** **sql_strm*, **stringstream** **select_strm*, **stringstream** **from_strm*, **stringstream** **where_strm_0*, **stringstream** **where_strm_1*, **bool** **first_select*, **bool** **first_from*, **bool** **first_where*, *bitset* < `QUERY_TYPE_BITSET_SIZE` > **field_flags*, **string** **return_str*);

111. Show. [LDF 2006.10.31.]
As well as simply showing the **Query_Type** object, this function is meant to serve as an example of traversing it. This is somewhat complicated, because of all of the **Query_Nodes** it contains, and the complex way in which they're linked. [LDF 2006.10.31.]
The **Scanner_Node** *scanner_node* data member of the **Query_Type** object may already be non-null. However, *show* has an optional **Scanner_Node** argument, just in case it doesn't. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this function.

[LDF 2006.11.14.] Added the optional **int** *level_ctr* argument with default 0.

[LDF 2006.12.12.] Added code to account for additional fields that refer to tables in the PICA database (PICA_DB).

⟨ Declare **Query_Type** functions 82 ⟩ +≡
int *show*(**string** $s =$ `"‘Query_Type’:\n"`, **Scanner_Node** $sscanner\_node = 0$);

112.
⟨ Define **Query_Type** functions 83 ⟩ +≡
int Query_Type :: *show*(**string** *s*, **Scanner_Node** *sscanner_node*){ **stringstream** *temp_strm*;
if ($scanner\_node \equiv 0 \land sscanner\_node \neq 0$) $scanner\_node = sscanner\_node$;

113. *query_type*, *field_type*, *value_type*, and *target_type*. [LDF 2006.10.31.]

Log

[LDF 2006.11.13.] Added code for showing *target_types*.

[LDF 2006.11.21.] Now showing *name*.

[LDF 2006.12.12.] Added code to account for **static const unsigned** *shorts* that refer to additional tables in the OAI database (dc_test).

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
temp_strm ≪ "‘name’␣==␣" ≪ name ≪ endl ≪ "‘query_type’␣==␣" ≪ query_type_map[query_type] ≪
    endl ≪ "‘field_type’␣==␣" ≪ field_type_map[field_type] ≪ endl ≪ "‘value_type’␣==␣" ≪
    value_type_map[value_type] ≪ endl ≪ "‘match_value’␣==␣" ≪ match_value_map[match_value] ≪
    endl;
int i = 0;
for (vector⟨unsigned short⟩ :: iterator iter = target_types.begin( ); iter ≠ target_types.end( ); ++iter)
  temp_strm ≪ "‘target_types[" ≪ i++ ≪ "]’␣==␣" ≪ target_type_map[*iter] ≪ endl;
if (value ≠ 0) {
  if (field_type ≡ ACCESS_NUMBER_FIELD)
    temp_strm ≪ "Access_Number␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ AUTHOR_FIELD)
    temp_strm ≪ "Author␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ BIBLIOGRAPHIC_TYPE_FIELD)
    temp_strm ≪ "Bibliographic_Type␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ CALL_NUMBER_FIELD)
    temp_strm ≪ "Call_Number␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ CLASSIFICATION_FIELD)
    temp_strm ≪ "Classification␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ COMPANY_FIELD)
    temp_strm ≪ "Company␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ CONTENT_SUMMARY_FIELD)
    temp_strm ≪ "Content_Summary␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ CONTRIBUTOR_FIELD)
    temp_strm ≪ "Contributor␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ CREATOR_FIELD)
    temp_strm ≪ "Creator␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ DATABASE_PROVIDER_FIELD)
    temp_strm ≪ "Database_Provider␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ DESCRIPTION_FIELD)
    temp_strm ≪ "Description␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ EXEMPLAR_PRODUCTION_NUMBER_FIELD)
    temp_strm ≪ "Exemplar_Production_Number␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ IDENTIFIER_FIELD)
    temp_strm ≪ "Identifier␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ INSTITUTION_FIELD)
    temp_strm ≪ "Institution␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ LANGUAGE_FIELD)
    temp_strm ≪ "Language␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
  else if (field_type ≡ MAIN_CANONICAL_TITLE_FIELD)
    temp_strm ≪ "Main_Canonical_Title_␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
```

```
else if (field_type ≡ PERMUTATION_PATTERN_FIELD)
  temp_strm ≪ "Permutation_Pattern␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
else if (field_type ≡ PERSON_FIELD)
  temp_strm ≪ "Person␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
else if (field_type ≡ PHYSICAL_DESCRIPTION_FIELD)
  temp_strm ≪ "Physical_Description␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
else if (field_type ≡ PUBLISHER_FIELD)
  temp_strm ≪ "Publisher␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_FIELD)
  temp_strm ≪ "Record␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_ID_FIELD)
  temp_strm ≪ "Record␣ID␣==␣" ≪ *static_cast⟨int *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_ELN_ORIGINAL_ENTRY_FIELD)
  temp_strm ≪ "Record_Eln_Original_Entry_Field␣==␣" ≪ *static_cast⟨int *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_ELN_MOST_RECENT_CHANGE_FIELD)
  temp_strm ≪ "Record_Eln_Most_Recent_Change_Field␣==␣" ≪ *static_cast⟨int
      *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_ELN_STATUS_CHANGE_FIELD)
  temp_strm ≪ "Record_Eln_Status_Change_Field␣==␣" ≪ *static_cast⟨int *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_IDENTIFICATION_NUMBER_FIELD)
  temp_strm ≪ "Record_Identification_Number_Field␣==␣" ≪ *static_cast⟨string
      *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_SOURCE_ID_FIELD)
  temp_strm ≪ "Record_Source_Id_Field␣==␣" ≪ *static_cast⟨int *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_YEAR_APPEARANCE_BEGIN_FIELD)
  temp_strm ≪ "Record_Year_Appearance_Begin_Field␣==␣" ≪ *static_cast⟨int *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_YEAR_APPEARANCE_END_FIELD)
  temp_strm ≪ "Record_Year_Appearance_End_Field␣==␣" ≪ *static_cast⟨int *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_YEAR_APPEARANCE_RAK_WB_FIELD)
  temp_strm ≪ "Record_Year_Appearance_Rak_Wb_Field␣==␣" ≪ *static_cast⟨int
      *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_YEAR_APPEARANCE_ORIGINAL_FIELD)
  temp_strm ≪ "Record_Year_Appearance_Original_Field␣==␣" ≪ *static_cast⟨int
      *⟩(value) ≪ endl;
else if (field_type ≡ RECORD_DATE_STATUS_CHANGE_FIELD)
  temp_strm ≪ "Record_Date_Status_Change_Field:" ≪ endl ≪
      *static_cast⟨Date_Time_Node⟩(value) ≪ endl;
else if (field_type ≡ RECORD_DATE_ORIGINAL_ENTRY_FIELD)
  temp_strm ≪ "Record_Date_Original_Entry_Field:" ≪ endl ≪
      *static_cast⟨Date_Time_Node⟩(value) ≪ endl;
else if (field_type ≡ RECORD_DATE_MOST_RECENT_CHANGE_FIELD)
  temp_strm ≪ "Record_Date_Most_Recent_Change_Field␣==␣" ≪ endl ≪
      *static_cast⟨Date_Time_Node⟩(value) ≪ endl;
else if (field_type ≡ REMOTE_ACCESS_FIELD)
  temp_strm ≪ "Remote_Access␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
else if (field_type ≡ RIGHTS_FIELD)
  temp_strm ≪ "Rights␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
else if (field_type ≡ SOURCE_FIELD)
  temp_strm ≪ "Source␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
else if (field_type ≡ SUBJECT_FIELD)
  temp_strm ≪ "Subject␣==␣" ≪ *static_cast⟨string *⟩(value) ≪ endl;
```

```
    else if (field_type ≡ SUPERORDINATE_ENTITIES_FIELD)
      temp_strm << "Superordinate_Entities␣==␣" << *static_cast⟨string *⟩(value) << endl;
    else if (field_type ≡ TITLE_FIELD)
      temp_strm << "Title␣==␣" << *static_cast⟨string *⟩(value) << endl;
    else if (field_type ≡ TYPE_FIELD)
      temp_strm << "Type␣==␣" << *static_cast⟨string *⟩(value) << endl;
    else temp_strm << "Can't␣show␣this␣field␣yet." << endl;
  }    /* if (value ≠ 0) */
  else {
    temp_strm << "`value'␣␣==␣0." << endl;
  }
  if (up ∧ up→value ∧ up→value_type ≡ QUERY_TYPE_STRING_TYPE) {
    temp_strm << "`up->value'␣==␣" << *static_cast⟨string *⟩(up→value) << endl;
  }
```

114. Output the contents of *temp_strm*. [LDF 2006.10.31.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  temp_strm.str("");
  if (scanner_node ≠ 0 ∧ scanner_node→log_strm.is_open()) scanner_node→log_strm << temp_strm;
```

115. Show subsidiary nodes recursively. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this section.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
if (and_node ≠ 0) {
   temp_strm ≪ "***␣‘and_node’␣****" ≪ endl;
   cerr_mutex.Lock();
   cerr ≪ temp_strm.str();
   cerr_mutex.Unlock();
   temp_strm.str("");
   if (scanner_node ≠ 0 ∧ scanner_node→log_strm.is_open()) scanner_node→log_strm ≪ temp_strm;
   and_node→show("‘and_node’:", sscanner_node);
}
if (or_node ≠ 0) {
   temp_strm ≪ "***␣‘or_node’␣****" ≪ endl;
   cerr_mutex.Lock();
   cerr ≪ temp_strm.str();
   cerr_mutex.Unlock();
   temp_strm.str("");
   if (scanner_node ≠ 0 ∧ scanner_node→log_strm.is_open()) scanner_node→log_strm ≪ temp_strm;
   or_node→show("‘or_node’:", sscanner_node);
}
if (xor_node ≠ 0) {
   temp_strm ≪ "***␣‘xor_node’␣****" ≪ endl;
   cerr_mutex.Lock();
   cerr ≪ temp_strm.str();
   cerr_mutex.Unlock();
   temp_strm.str("");
   if (scanner_node ≠ 0 ∧ scanner_node→log_strm.is_open()) scanner_node→log_strm ≪ temp_strm;
   xor_node→show("‘xor_node’:", sscanner_node);
}
temp_strm ≪ "*******" ≪ endl;
cerr_mutex.Lock();
cerr ≪ temp_strm.str();
cerr_mutex.Unlock();
temp_strm.str("");
```

116.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
return 0; }     /* End of Query_Type::show definition. */
```

117. Putting Query_Type together. [LDF 2006.10.31.]

118. This is what's compiled.

⟨ Include files 10 ⟩
⟨ `querytyp.web` 45 ⟩

using namespace std;
using namespace Scan_Parse;

⟨ Preprocessor macro definitions 73 ⟩
⟨ Forward declarations 13 ⟩
⟨ Declare **class Query_Type** 49 ⟩
⟨ Initialize **static Query_Type** constants 54 ⟩
⟨ Initialize **static Query_Type** variables 76 ⟩
⟨ Define **Query_Type** functions 83 ⟩

119. This is what's written to `querytyp.h`.

```
⟨ querytyp.hh 119 ⟩ ≡
#ifndef QUERYTYP_KNOWN
#define QUERYTYP_KNOWN 1
  ⟨ Preprocessor macro definitions 73 ⟩
  ⟨ Preprocessor macro calls 19 ⟩
  using namespace std;
  ⟨ Forward declarations 13 ⟩
  ⟨ Declare class Query_Type 49 ⟩
#endif    /* QUERYTYP_KNOWN is defined. */
```

120. Query_Type::*generate_sql_string* definition (`qtgensql.web`). [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Created this file.

```
⟨ qtgensql.web 120 ⟩ ≡
  static char id_string[] = "$Id:␣qtgensql.web,v␣1.4␣2007/01/02␣12:20:53␣lfinsto1␣Exp␣$";
```

This code is cited in sections 6 and 8.

This code is used in section 154.

121. Include files.

```
⟨ Include files 10 ⟩ +≡
#include "localldf.h"
#ifdef NON_WIN_LDF
#include "nonwin.h"
#else
#include "stdafx.h"
#endif
#include "parser.hxx"
#include "dttmtype.h"
#include "idtype.h"
#include "scnrtype.h"
#include "scanner.h"
#include "querytyp.h"
```

122. Generate SQL string. [LDF 2006.12.01.]

Log

[LDF 2006.12.01.] Added this function.

[LDF 2006.12.12.] Added code for `MAIN_CANONICAL_TITLE_FLAG`.

[LDF 2006.12.12.] Added code to account for **static const unsigned** *shorts* that refer to additional tables in the OAI database (dc_test).

[LDF 2006.12.14.] Moved the definition of this function from `querytyp.web` to this file (`qtgensql.web`).

123.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
int Query_Type::generate_sql_string(Scanner_Node scanner_node, int database_type, stringstream
        *sql_strm, stringstream *select_strm, stringstream *from_strm, stringstream
        *where_strm_0, stringstream *where_strm_1, bool *first_select, bool *first_from, bool
        *first_where, bitset < QUERY_TYPE_BITSET_SIZE > *field_flags, string *return_str){
#if 0    /* 14g */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
    stringstream date_strm;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣‘Query_Type::generate_sql_string’." << endl <<
        "‘query_ctr’␣==␣" << query_ctr << endl << "‘database_type’␣==␣" <<
        token_map[database_type] << endl << "‘query_type’␣==␣" << query_type_map[query_type] <<
        endl << "‘*first_select’␣==␣" << *first_select << endl << "‘*first_where’␣==␣" <<
        *first_where << endl << "‘*first_from␣==␣" << *first_from << endl << "‘query_ctr’␣==␣" <<
        query_ctr << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (scanner_node ≠ 0) scanner_node→log_strm << temp_strm.str();
    temp_strm.str("");
    getchar();
#endif
```

124. Set match operator string. [LDF 2006.12.12.]

Log

[2006.12.12.] Added this declaration.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
  string match_str;
  if (match_value ≡ QUERY_TYPE_NULL_TYPE) match_str = "=";
  else if (match_value ≡ CONTAINS_VALUE) match_str = "contains";
  else if (match_value ≡ FREETEXT_VALUE) match_str = "freetext";
  else if (match_value ≡ LIKE_VALUE) match_str = "like";
  else {
    temp_strm << "WARNING!␣␣In␣‘Query_Type::generate_sql_string’:" << endl <<
        "‘match_value’␣has␣invalid␣value:␣␣" << match_value << endl <<
        "Will␣use␣‘=’␣as␣the␣match␣operator." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (scanner_node ≠ 0) scanner_node→log_strm << temp_strm.str();
    temp_strm.str("");
  }
```

125. Set *query_type_str*. [LDF 2006.12.06.]

⟨ Define **Query_Type** functions 83 ⟩ +≡
```
string query_type_str;
```

126. Set parenthesis strings. [LDF 2006.12.14.]

⟨ Define **Query_Type** functions 83 ⟩ +≡
```
string open_parenthesis_str;
string close_parenthesis_str;
if (and_node ∨ or_node ∨ xor_node) {
  open_parenthesis_str = "(";
  close_parenthesis_str = ")";
}
else {
  open_parenthesis_str = "";
  close_parenthesis_str = "";
}
if (query_type ≡ AND_TYPE) query_type_str = "and";
else if (query_type ≡ OR_TYPE) query_type_str = "or";
else if (query_type ≡ XOR_TYPE) {
  if (database_type ≡ OAI) {
    temp_strm ≪ "ERROR!␣␣In␣‘Query_Type::generate_sql_string’:" ≪ endl ≪
        "‘query_type’␣==␣‘XOR_TYPE’␣and␣‘database_type’␣==␣‘OAI’." ≪
        endl ≪ "This␣combination␣of␣types␣is␣not␣allowed." ≪ endl ≪
        "Exiting␣function␣unsuccessfully␣with␣return␣value␣1." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    *return_str = "";
    return 1;
  }    /* if (database_type ≡ OAI) */
  query_type_str = "xor";
}    /* else if (query_type ≡ XOR_TYPE) */
if (negated) query_type_str += "␣not";
```

127.

⟨ Define **Query_Type** functions 83 ⟩ +≡
```
  if (query_ctr ≡ 0) *select_strm ≪ "select␣" ≪ endl;
#if 0    /* 1 */
  cerr ≪ "‘field_type’␣==␣" ≪ field_type_map[field_type] ≪ endl;
#endif
```

128. AUTHOR_SURNAME_FIELD. Pica Only. [LDF 2006.12.06.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
if (field_type ≡ AUTHOR_SURNAME_FIELD ∧ database_type ≡ PICA) {
  if (*first_select) {
    *select_strm ≪ "RAUT.record_id";
    *first_select = false;
  }
  if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
    *select_strm ≪ ",␣AUT.author_id" ≪ endl;
  }
  if ((*field_flags & AUTHOR_SURNAME_FLAG) ≡ NULL_BITSET) {
    *select_strm ≪ ",␣AUT.author_surname" ≪ endl;
  }
  if (*first_from) *from_strm ≪ "from␣";
  else if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
    *from_strm ≪ ",␣";
  }
  *first_from = false;
  if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
    *from_strm ≪ "Authors␣as␣AUT,␣Records_Authors␣as␣RAUT" ≪ endl;
  }
  if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
  else {
    *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
  }
  *first_where = false;
  *where_strm_0 ≪ "AUT.author_surname␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string
      *⟩(value) ≪ "'␣";
  if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
    *where_strm_1 ≪ "and␣RAUT.author_id␣=␣AUT.author_id" ≪ endl;
  }
  *field_flags |= AUTHOR_SURNAME_FLAG;
  *field_flags |= AUTHOR_FLAG;
}   /* if (field_type ≡ AUTHOR_SURNAME_FIELD ∧ database_type ≡ PICA) */
```

129. AUTHOR_GIVEN_NAME_FIELD. Pica Only. [LDF 2006.12.13.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
else
  if (field_type ≡ AUTHOR_GIVEN_NAME_FIELD ∧ database_type ≡ PICA) {
    if (*first_select) {
      *select_strm ≪ "RAUT.record_id";
      *first_select = false;
    }
    if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
      *select_strm ≪ ",␣AUT.author_id" ≪ endl;
    }
    if ((*field_flags & AUTHOR_GIVEN_NAME_FLAG) ≡ NULL_BITSET) {
      *select_strm ≪ ",␣AUT.author_given_name" ≪ endl;
    }
    if (*first_from) *from_strm ≪ "from␣";
    else if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ ",␣";
    }
    *first_from = false;
    if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ "Authors␣as␣AUT,␣Records_Authors␣as␣RAUT" ≪ endl;
    }
    if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
    else {
      *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 ≪ "AUT.author_given_name␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string
        *⟩(value) ≪ "'␣";
    if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
      *where_strm_1 ≪ "and␣RAUT.author_id␣=␣AUT.author_id" ≪ endl;
    }
    *field_flags |= AUTHOR_GIVEN_NAME_FLAG;
    *field_flags |= AUTHOR_FLAG;
  }     /* else if (field_type ≡ AUTHOR_GIVEN_NAME_FIELD ∧ database_type ≡ PICA) */
```

130. `AUTHOR_PREFIX_FIELD`. Pica Only. [LDF 2006.12.13.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
else
  if (field_type ≡ AUTHOR_PREFIX_FIELD ∧ database_type ≡ PICA) {
    if (*first_select) {
      *select_strm ≪ "RAUT.record_id";
      *first_select = false;
    }
    if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
      *select_strm ≪ ",␣AUT.author_id" ≪ endl;
    }
    if ((*field_flags & AUTHOR_PREFIX_FLAG) ≡ NULL_BITSET) {
      *select_strm ≪ ",␣AUT.author_prefix" ≪ endl;
    }
    if (*first_from) *from_strm ≪ "from␣";
    else if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ ",␣";
    }
    *first_from = false;
    if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ "Authors␣as␣AUT,␣Records_Authors␣as␣RAUT" ≪ endl;
    }
    if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
    else {
      *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 ≪ "AUT.author_prefix␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string
        *⟩(value) ≪ "'␣";
    if ((*field_flags & AUTHOR_FLAG) ≡ NULL_BITSET) {
      *where_strm_1 ≪ "and␣RAUT.author_id␣=␣AUT.author_id" ≪ endl;
    }
    *field_flags |= AUTHOR_PREFIX_FLAG;
    *field_flags |= AUTHOR_FLAG;
  }    /* else if (field_type ≡ AUTHOR_PREFIX_FIELD ∧ database_type ≡ PICA) */
```

131. `CONTRIBUTOR_SURNAME_FIELD`. Pica Only. [LDF 2006.12.14.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
if (field_type ≡ CONTRIBUTOR_SURNAME_FIELD ∧ database_type ≡ PICA) {
  if (*first_select) {
    *select_strm ≪ "RCNTRB.record_id";
    *first_select = false;
  }
  if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
    *select_strm ≪ ",␣CNTRB.contributor_id" ≪ endl;
  }
  if ((*field_flags & CONTRIBUTOR_SURNAME_FLAG) ≡ NULL_BITSET) {
    *select_strm ≪ ",␣CNTRB.contributor_surname" ≪ endl;
  }
  if (*first_from) *from_strm ≪ "from␣";
  else if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
    *from_strm ≪ ",␣";
  }
  *first_from = false;
  if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
    *from_strm ≪ "Contributors␣as␣CNTRB,␣Records_Contributors␣as␣RCNTRB" ≪ endl;
  }
  if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
  else {
    *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
  }
  *first_where = false;
  *where_strm_0 ≪ "CNTRB.contributor_surname␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string
      *⟩(value) ≪ "'␣";
  if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
    *where_strm_1 ≪ "and␣RCNTRB.contributor_id␣=␣CNTRB.contributor_id" ≪ endl;
  }
  *field_flags |= CONTRIBUTOR_SURNAME_FLAG;
  *field_flags |= CONTRIBUTOR_FLAG;
}   /* if (field_type ≡ CONTRIBUTOR_SURNAME_FIELD ∧ database_type ≡ PICA) */
```

132. CONTRIBUTOR_GIVEN_NAME_FIELD. Pica Only. [LDF 2006.12.14.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
else
  if (field_type ≡ CONTRIBUTOR_GIVEN_NAME_FIELD ∧ database_type ≡ PICA) {
    if (*first_select) {
      *select_strm ≪ "RCNTRB.record_id";
      *first_select = false;
    }
    if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
      *select_strm ≪ ",␣CNTRB.contributor_id" ≪ endl;
    }
    if ((*field_flags & CONTRIBUTOR_GIVEN_NAME_FLAG) ≡ NULL_BITSET) {
      *select_strm ≪ ",␣CNTRB.contributor_given_name" ≪ endl;
    }
    if (*first_from) *from_strm ≪ "from␣";
    else if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ ",␣";
    }
    *first_from = false;
    if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ "Contributors␣as␣CNTRB,␣Records_Contributors␣as␣RCNTRB" ≪ endl;
    }
    if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
    else {
      *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 ≪ "CNTRB.contributor_given_name␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string
        *⟩(value) ≪ "'␣";
    if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
      *where_strm_1 ≪ "and␣RCNTRB.contributor_id␣=␣CNTRB.contributor_id" ≪ endl;
    }
    *field_flags |= CONTRIBUTOR_GIVEN_NAME_FLAG;
    *field_flags |= CONTRIBUTOR_FLAG;
  }   /* else if (field_type ≡ CONTRIBUTOR_GIVEN_NAME_FIELD ∧ database_type ≡ PICA) */
```

133. CONTRIBUTOR_PREFIX_FIELD. Pica Only. [LDF 2006.12.14.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
else
  if (field_type ≡ CONTRIBUTOR_PREFIX_FIELD ∧ database_type ≡ PICA) {
    if (*first_select) {
      *select_strm ≪ "RCNTRB.record_id";
      *first_select = false;
    }
    if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
      *select_strm ≪ ",␣CNTRB.contributor_id" ≪ endl;
    }
    if ((*field_flags & CONTRIBUTOR_PREFIX_FLAG) ≡ NULL_BITSET) {
      *select_strm ≪ ",␣CNTRB.contributor_prefix" ≪ endl;
    }
    if (*first_from) *from_strm ≪ "from␣";
    else if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ ",␣";
    }
    *first_from = false;
    if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ "Contributors␣as␣CNTRB,␣Records_Contributors␣as␣RCNTRB" ≪ endl;
    }
    if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
    else {
      *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 ≪ "CNTRB.contributor_prefix␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string
        *⟩(value) ≪ "'␣";
    if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) {
      *where_strm_1 ≪ "and␣RCNTRB.contributor_id␣=␣CNTRB.contributor_id" ≪ endl;
    }
    *field_flags |= CONTRIBUTOR_PREFIX_FLAG;
    *field_flags |= CONTRIBUTOR_FLAG;
  }    /* else if (field_type ≡ CONTRIBUTOR_PREFIX_FIELD ∧ database_type ≡ PICA) */
```

134. OAI. [LDF 2006.12.06.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
else
  if (field_type ≡ CONTRIBUTOR_FIELD ∧ database_type ≡ OAI) {
    if (*first_select) {
      *select_strm ≪ "RCN.record_id";
      *first_select = false;
    }
    if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET)
      *select_strm ≪ ",␣CN.contributor_id,␣CN.dc_contributor" ≪ endl;
    if (*first_from) *from_strm ≪ "from␣";
    else if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET) *from_strm ≪ ",␣";
    *first_from = false;
    if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET)
      *from_strm ≪ "Contributors␣as␣CN,␣Records_Contributors␣as␣RCN" ≪ endl;
    if (*first_where) *where_strm_0 ≪ "where␣";
    else {
      *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 ≪ "CN.dc_contributor␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string
        *⟩(value) ≪ "'␣";
    if ((*field_flags & CONTRIBUTOR_FLAG) ≡ NULL_BITSET)
      *where_strm_1 ≪ "and␣RCN.contributor_id␣=␣CN.contributor_id" ≪ endl;
    *field_flags |= CONTRIBUTOR_FLAG;
  }    /* else if (field_type ≡ CONTRIBUTOR_FIELD ∧ database_type ≡ OAI) */
```

135. Creator. OAI only. [LDF 2006.12.06.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
else
if (field_type ≡ CREATOR_FIELD ∧ database_type ≡ OAI) {
  if (*first_select) {
    *select_strm ≪ "RC.record_id";
    *first_select = false;
  }
  if ((*field_flags & CREATOR_FLAG) ≡ NULL_BITSET)
    *select_strm ≪ ",␣C.creator_id,␣C.dc_creator" ≪ endl;
  if (*first_from) *from_strm ≪ "from␣";
  else if ((*field_flags & CREATOR_FLAG) ≡ NULL_BITSET) *from_strm ≪ ",␣";
  *first_from = false;
  if ((*field_flags & CREATOR_FLAG) ≡ NULL_BITSET)
    *from_strm ≪ "Creators␣as␣C,␣Records_Creators␣as␣RC" ≪ endl;
  if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
  else {
    *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
  }
  *first_where = false;
  *where_strm_0 ≪ "C.dc_creator␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string *⟩(value) ≪ "'␣";
  if ((*field_flags & CREATOR_FLAG) ≡ NULL_BITSET)
    *where_strm_1 ≪ "and␣RC.creator_id␣=␣C.creator_id" ≪ endl;
  *field_flags |= CREATOR_FLAG;
}    /* else if (field_type ≡ CREATOR_FIELD ∧ database_type ≡ OAI) */
else if (field_type ≡ TITLE_FIELD) {
```

136. OAI. [LDF 2006.12.12.]

Log

[2006.12.11.] Added this section.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  if (database_type ≡ OAI) {
    if (*first_select) {
      *select_strm ≪ "T.record_id";
      *first_select = false;
    }
    if ((*field_flags & TITLE_FLAG) ≡ NULL_BITSET)
      *select_strm ≪ ",␣T.title_id,␣T.dc_title" ≪ endl;
    if (*first_from) *from_strm ≪ "from␣";
    else if ((*field_flags & TITLE_FLAG) ≡ NULL_BITSET) *from_strm ≪ ",␣";
    *first_from = false;
    if ((*field_flags & TITLE_FLAG) ≡ NULL_BITSET)
      *from_strm ≪ "Titles␣as␣T,␣Records␣as␣REC" ≪ endl;
    if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
    else {
      *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 ≪ "T.dc_title␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string *⟩(value) ≪ "'␣";
    if ((*field_flags & TITLE_FLAG) ≡ NULL_BITSET)
      *where_strm_1 ≪ "and␣T.record_id␣=␣REC.record_id" ≪ endl;
    *field_flags |= TITLE_FLAG;
  }     /* if (database_type ≡ OAI) */
  }     /* else if (field_type ≡ TITLE_FIELD) */
```

137. Main canonical title (Pica). [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this section.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  else if (field_type ≡ MAIN_CANONICAL_TITLE_FIELD) {
```

138. PICA. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this section.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  if (database_type ≡ PICA) {
    if (*first_select) {
      *select_strm ≪ "RMT.record_id";
      *first_select = false;
    }
    if ((*field_flags & MAIN_CANONICAL_TITLE_FLAG) ≡ NULL_BITSET)
      *select_strm ≪ ",␣MT.main_title_id,␣MT.main_canonical_title" ≪ endl;
    if (*first_from) *from_strm ≪ "from␣";
    else if ((*field_flags & MAIN_CANONICAL_TITLE_FLAG) ≡ NULL_BITSET) *from_strm ≪ ",␣";
    *first_from = false;
    if ((*field_flags & MAIN_CANONICAL_TITLE_FLAG) ≡ NULL_BITSET)
      *from_strm ≪ "Main_Titles␣as␣MT,␣Records_Main_Titles␣as␣RMT" ≪ endl;
    if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
    else {
      *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 ≪ "MT.main_canonical_title␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string
        *⟩(value) ≪ "'␣";
    if ((*field_flags & MAIN_CANONICAL_TITLE_FLAG) ≡ NULL_BITSET)
      *where_strm_1 ≪ "and␣RMT.main_title_id␣=␣MT.main_title_id" ≪ endl;
    *field_flags |= MAIN_CANONICAL_TITLE_FLAG;
  }    /* if (database_type ≡ PICA) */
  }    /* else if (field_type ≡ MAIN_CANONICAL_TITLE_FIELD) */
```

139. Record. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

---

140. Record ID. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  else
    if (field_type ≡ RECORD_ID_FIELD ∧ database_type ≡ PICA) {
      if (*first_select) {
        *select_strm ≪ "REC.record_id" ≪ endl;
        *first_select = false;
      }
      else if ((*field_flags & RECORD_ID_FLAG) ≡ NULL_BITSET) {
        *select_strm ≪ ",␣REC.record_id" ≪ endl;
      }
      if (*first_from) *from_strm ≪ "from␣";
      else if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm ≪ ",␣";
      }
      *first_from = false;
      if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm ≪ "Records␣as␣REC" ≪ endl;
      }
      if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
      else {
        *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
      }
      *first_where = false;
      *where_strm_0 ≪ "REC.record_id␣" ≪ match_str ≪ "␣" ≪ *static_cast⟨int *⟩(value) ≪ "␣";
      *field_flags |= RECORD_ID_FLAG;
      *field_flags |= RECORD_FLAG;
    }    /* else if (field_type ≡ RECORD_ID_FIELD ∧ database_type ≡ PICA) */
```

141. Record ELN Original Entry. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  else
    if (field_type ≡ RECORD_ELN_ORIGINAL_ENTRY_FIELD ∧ database_type ≡ PICA) {
      if (*first_select) {
        *select_strm ≪ "REC.record_id";
        *first_select = false;
      }
      if ((*field_flags & RECORD_ELN_ORIGINAL_ENTRY_FLAG) ≡ NULL_BITSET) {
        *select_strm ≪ ",␣REC.eln_original_entry" ≪ endl;
      }
      if (*first_from) *from_strm ≪ "from␣";
      else if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm ≪ ",␣";
      }
      *first_from = false;
      if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm ≪ "Records␣as␣REC" ≪ endl;
      }
      if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
      else {
        *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
      }
      *first_where = false;
      *where_strm_0 ≪ "REC.eln_original_entry␣" ≪ match_str ≪ "␣" ≪ *static_cast⟨int
          *⟩(value) ≪ "␣";
      *field_flags |= RECORD_ELN_ORIGINAL_ENTRY_FLAG;
      *field_flags |= RECORD_FLAG;
    }    /* else if (field_type ≡ RECORD_ELN_ORIGINAL_ENTRY_FIELD ∧ database_type ≡ PICA) */
```

142. Record ELN Most Recent Change. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  else
    if (field_type ≡ RECORD_ELN_MOST_RECENT_CHANGE_FIELD ∧ database_type ≡ PICA) {
      if (*first_select) {
        *select_strm << "REC.record_id";
        *first_select = false;
      }
      if ((*field_flags & RECORD_ELN_MOST_RECENT_CHANGE_FLAG) ≡ NULL_BITSET) {
        *select_strm << ",␣REC.eln_most_recent_change" << endl;
      }
      if (*first_from) *from_strm << "from␣";
      else if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm << ",␣";
      }
      *first_from = false;
      if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm << "Records␣as␣REC" << endl;
      }
      if (*first_where) *where_strm_0 << "where␣" << endl;
      else {
        *where_strm_0 << query_type_str << "␣" << open_parenthesis_str;
      }
      *first_where = false;
      *where_strm_0 << "REC.eln_most_recent_change␣" << match_str << "␣" << *static_cast⟨int
          *⟩(value) << "␣";
      *field_flags |= RECORD_ELN_MOST_RECENT_CHANGE_FLAG;
      *field_flags |= RECORD_FLAG;
    }    /* else if (field_type ≡ RECORD_ELN_MOST_RECENT_CHANGE_FIELD ∧ database_type ≡ PICA) */
```

143. Record ELN Status Change. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this section.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
else
  if (field_type ≡ RECORD_ELN_STATUS_CHANGE_FIELD ∧ database_type ≡ PICA) {
    if (*first_select) {
      *select_strm ≪ "REC.record_id";
      *first_select = false;
    }
    if ((*field_flags & RECORD_ELN_STATUS_CHANGE_FLAG) ≡ NULL_BITSET) {
      *select_strm ≪ ",␣REC.eln_status_change" ≪ endl;
    }
    if (*first_from) *from_strm ≪ "from␣";
    else if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ ",␣";
    }
    *first_from = false;
    if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
      *from_strm ≪ "Records␣as␣REC" ≪ endl;
    }
    if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
    else {
      *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 ≪ "REC.eln_status_change␣" ≪ match_str ≪ "␣" ≪ *static_cast⟨int
        *⟩(value) ≪ "␣";
    *field_flags |= RECORD_ELN_STATUS_CHANGE_FLAG;
    *field_flags |= RECORD_FLAG;
  }   /* else if (field_type ≡ RECORD_ELN_STATUS_CHANGE_FIELD ∧ database_type ≡ PICA) */
```

144. Record Identification Number. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this section.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  else
    if (field_type ≡ RECORD_IDENTIFICATION_NUMBER_FIELD ∧ database_type ≡ PICA) {
      if (*first_select) {
        *select_strm ≪ "REC.record_id";
        *first_select = false;
      }
      if ((*field_flags & RECORD_IDENTIFICATION_NUMBER_FLAG) ≡ NULL_BITSET) {
        *select_strm ≪ ",␣REC.identification_number" ≪ endl;
      }
      if (*first_from) *from_strm ≪ "from␣";
      else if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm ≪ ",␣";
      }
      *first_from = false;
      if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm ≪ "Records␣as␣REC" ≪ endl;
      }
      if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
      else {
        *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
      }
      *first_where = false;
      *where_strm_0 ≪ "REC.identification_number␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string
          *⟩(value) ≪ "'␣";
      *field_flags |= RECORD_IDENTIFICATION_NUMBER_FLAG;
      *field_flags |= RECORD_FLAG;
    }     /* else if (field_type ≡ RECORD_IDENTIFICATION_NUMBER_FIELD ∧ database_type ≡ PICA) */
```

145. Record Date Status Change. [LDF 2006.12.19.]

Log

[LDF 2006.12.19.] Added this section.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  else
    if (field_type ≡ RECORD_DATE_STATUS_CHANGE_FIELD ∧ database_type ≡ PICA) {
      if (*first_select) {
        *select_strm ≪ "REC.record_id";
        *first_select = false;
      }
      if ((*field_flags & RECORD_DATE_STATUS_CHANGE_FLAG) ≡ NULL_BITSET) {
        *select_strm ≪ ",␣REC.date_status_change" ≪ endl;
      }
      if (*first_from) *from_strm ≪ "from␣";
      else if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm ≪ ",␣";
      }
      *first_from = false;
      if ((*field_flags & RECORD_FLAG) ≡ NULL_BITSET) {
        *from_strm ≪ "Records␣as␣REC" ≪ endl;
      }
      if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
      else {
        *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
      }
      *first_where = false;
      Date_Time_Node d = static_cast⟨Date_Time_Node⟩(value);
      date_strm.str("");
      *where_strm_0 ≪ "REC.date_status_change␣" ≪ match_str ≪ "␣convert(datetime,␣'";
      if (d→year) *where_strm_0 ≪ setfill('0') ≪ right ≪ setw(4) ≪ *d→year;
      else *where_strm_0 ≪ "0000";
      *where_strm_0 ≪ "-";
      if (d→month) *where_strm_0 ≪ setfill('0') ≪ right ≪ setw(2) ≪ *d→month;
      else *where_strm_0 ≪ "00";
      *where_strm_0 ≪ "-";
      if (d→day) *where_strm_0 ≪ setfill('0') ≪ right ≪ setw(2) ≪ *d→day;
      else *where_strm_0 ≪ "00";
      *where_strm_0 ≪ "␣";
      if (d→hour) *where_strm_0 ≪ setfill('0') ≪ right ≪ setw(2) ≪ *d→hour;
      else *where_strm_0 ≪ "00";
      *where_strm_0 ≪ ":";
      if (d→minute) *where_strm_0 ≪ setfill('0') ≪ right ≪ setw(2) ≪ *d→minute;
      else *where_strm_0 ≪ "00";
      *where_strm_0 ≪ ":";
      if (d→second) {
        if (*d→second < 10) *where_strm_0 ≪ "0";
        *where_strm_0 ≪ fixed ≪ setprecision(3) ≪ *d→second;
      }
      else *where_strm_0 ≪ "00.000";
```

```
    *where_strm_0 << "',␣121)";
    *field_flags |= RECORD_DATE_STATUS_CHANGE_FLAG;
    *field_flags |= RECORD_FLAG;
  }  /* else if (field_type ≡ RECORD_DATE_STATUS_CHANGE_FIELD ∧ database_type ≡ PICA) */
```

146. Subject. [LDF Undated.]

```
⟨ Define Query_Type functions 83 ⟩ +≡
  else if (field_type ≡ SUBJECT_FIELD) {
```

147.

Log

[2006.12.12.] Added this section.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  if (database_type ≡ PICA) {
    if (*first_select) {
      *select_strm << "RS.record_id";
      *first_select = false;
    }
    if ((*field_flags & SUBJECT_FLAG) ≡ NULL_BITSET)
      *select_strm << ",␣S.subject_id,␣S.subject" << endl;
    if (*first_from) *from_strm << "from␣";
    else if ((*field_flags & SUBJECT_FLAG) ≡ NULL_BITSET) *from_strm << ",␣";
    *first_from = false;
    if ((*field_flags & SUBJECT_FLAG) ≡ NULL_BITSET)
      *from_strm << "Subjects␣as␣S,␣Records_Subjects␣as␣RS" << endl;
    if (*first_where) *where_strm_0 << "where␣" << endl;
    else {
      *where_strm_0 << query_type_str << "␣" << open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 << "S.subject␣" << match_str << "␣'" << *static_cast⟨string *⟩(value) << "'␣";
    if ((*field_flags & SUBJECT_FLAG) ≡ NULL_BITSET)
      *where_strm_1 << "and␣RS.subject_id␣=␣S.subject_id" << endl;
    *field_flags |= SUBJECT_FLAG;
  }  /* if (database_type ≡ PICA) */
```

148.

```
⟨ Define Query_Type functions 83 ⟩ +≡
  if (database_type ≡ OAI) {
    if (*first_select) {
      *select_strm ≪ "RS.record_id";
      *first_select = false;
    }
    if ((*field_flags & SUBJECT_FLAG) ≡ NULL_BITSET)
      *select_strm ≪ ",␣S.subject_id,␣S.dc_subject" ≪ endl;
    if (*first_from) *from_strm ≪ "from␣";
    else if ((*field_flags & SUBJECT_FLAG) ≡ NULL_BITSET) *from_strm ≪ ",␣";
    *first_from = false;
    if ((*field_flags & SUBJECT_FLAG) ≡ NULL_BITSET)
      *from_strm ≪ "Subjects␣as␣S,␣Records_Subjects␣as␣RS" ≪ endl;
    if (*first_where) *where_strm_0 ≪ "where␣" ≪ endl;
    else {
      *where_strm_0 ≪ query_type_str ≪ "␣" ≪ open_parenthesis_str;
    }
    *first_where = false;
    *where_strm_0 ≪ "S.dc_subject␣" ≪ match_str ≪ "␣'" ≪ *static_cast⟨string *⟩(value) ≪ "'␣";
    if ((*field_flags & SUBJECT_FLAG) ≡ NULL_BITSET)
      *where_strm_1 ≪ "and␣RS.subject_id␣=␣S.subject_id" ≪ endl;
    *field_flags |= SUBJECT_FLAG;
  }   /* if (database_type ≡ OAI) */
  }   /* else if (field_type ≡ SUBJECT_FIELD) */
  temp_strm ≪ "`value_type'␣==␣" ≪ value_type_map[value_type] ≪ endl;
  temp_strm ≪ "`target_types'␣==␣";
  if (target_types.size() ≤ 0) temp_strm ≪ "QUERY_TYPE_NULL_TYPE";
  else if (target_types.size() ≡ 1) {
    temp_strm ≪ target_type_map[target_types[0]];
  }
  else {
    int i = 0;
    for (vector⟨unsigned short⟩::iterator iter = target_types.begin(); iter ≠ target_types.end();
          ++iter) {
      if (i++ > 0) temp_strm ≪ endl ≪ "--" ≪ endl;
      temp_strm ≪ target_type_map[*iter];
    }   /* for */
  }   /* else */
  temp_strm ≪ endl;
  if (value_type ≡ QUERY_TYPE_STRING_TYPE) {
    temp_strm ≪ *static_cast⟨string *⟩(value) ≪ endl;
    if (value_type ≡ QUERY_TYPE_STRING_TYPE) ;
#if 0   /* 1 */
    *sql_strm ≪ *static_cast⟨string *⟩(value) ≪ "␣";
#endif
  }
  if (up) temp_strm ≪ "`up'␣==␣" ≪ up→query_ctr;
  else temp_strm ≪ "`up'␣==␣0";
  temp_strm ≪ endl;
  if (and_node) {
    temp_strm ≪ "`and'␣==␣" ≪ and_node→query_ctr;
```

```
  }
  else temp_strm << "`and_node'␣==␣0";
  temp_strm << endl;
  if (or_node) {
    temp_strm << "`or'␣==␣" << or_node→query_ctr;
  }
  else temp_strm << "`or_node'␣==␣0";
  temp_strm << endl;
  if (or_node) {
    temp_strm << "`xor'␣==␣" << xor_node→query_ctr;
  }
  else temp_strm << "`xor_node'␣==␣0";
  temp_strm << endl;
  if (and_node) {
    temp_strm << "`and_node'␣==␣" << endl;
    *where_strm_0 << endl;
    and_node→generate_sql_string(scanner_node, database_type, sql_strm, select_strm, from_strm,
        where_strm_0, where_strm_1, first_select, first_from, first_where, field_flags, return_str);
  }
  if (or_node) {
    temp_strm << "`or_node'␣==␣" << endl;
    *where_strm_0 << endl;
    or_node→generate_sql_string(scanner_node, database_type, sql_strm, select_strm, from_strm,
        where_strm_0, where_strm_1, first_select, first_from, first_where, field_flags, return_str);
  }
  if (xor_node) {
    temp_strm << "`xor_node'␣==␣" << endl;
    *where_strm_0 << endl;
    xor_node→generate_sql_string(scanner_node, database_type, sql_strm, select_strm, from_strm,
        where_strm_0, where_strm_1, first_select, first_from, first_where, field_flags, return_str);
  }
```

149.

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
  if (query_ctr > 0) *where_strm_0 << close_parenthesis_str;
```

150. Write streams to **sql_strm*. [LDF 2006.12.05.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
  if (query_ctr ≡ 0) {
    *sql_strm << select_strm→str() << from_strm→str() << where_strm_0→str() << endl <<
        where_strm_1→str();
  }
```

151. Print debuggging output to terminal. [LDF 2006.12.05.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
  if (query_ctr ≡ 0) temp_strm << endl << "`sql_strm->str()'␣==␣" << endl << sql_strm→str() <<
      endl << "Enter␣<RETURN>␣to␣continue:␣";
```

152. Store string and return. [LDF 2006.12.01.]

⟨ Define **Query_Type** functions 83 ⟩ +≡

```
#ifdef DEBUG_OUTPUT
  temp_strm << "Exiting␣‘Query_Type::generate_sql_string’." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  if (scanner_node ≠ 0) scanner_node→log_strm << temp_strm.str();
  temp_strm.str("");
#endif
  *return_str = sql_strm→str();
  return 0; }    /* Query_Type::generate_sql_string */
```

153. Putting Query_Type::*generate_sql_string* together. [LDF 2006.10.31.]

154. This is what's compiled.

⟨ Include files 10 ⟩
⟨ `qtgensql.web` 120 ⟩
using namespace std;
using namespace Scan_Parse;
⟨ Define **Query_Type** functions 83 ⟩

155. datasource_type (`dtsrctyp.web`). [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Created this file.

⟨ `dtsrctyp.web` 155 ⟩ ≡
```
static char id_string[] = "$Id: dtsrctyp.web,v 1.5 2007/01/02 12:39:32 lfinsto1 Exp $";
```
This code is cited in sections 6 and 8.

This code is used in section 182.

156. Include files.

⟨ Include files 10 ⟩ +≡
```
#include "localldf.h"
#ifdef NON_WIN_LDF
#include "nonwin.h"
#else
#include "stdafx.h"
#endif
#include "parser.hxx"
#include "dttmtype.h"
#include "idtype.h"
#include "scnrtype.h"
#include "scanner.h"
```

157. Preprocessor macro calls.

⟨ Preprocessor macro calls 19 ⟩ +≡
```
#ifdef WIN_LDF
#pragma once
#endif
```

158. Forward declarations. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this section.

⟨ Forward declarations 13 ⟩ +≡
```
class Scanner_Type;
typedef Scanner_Type *Scanner_Node;
class Datasource_Type;
typedef Datasource_Type *Datasource_Node;
class Id_Type;
typedef Id_Type *Id_Node;
```

159. class datasource_type declaration. . [LDF 2006.12.08.]
The forward declaration of **Datasource_Type** is needed in order to be able to define **Datasource_Node** using **typedef**. [LDF 2006.12.08.]

Log

Added this section.

⟨ Declare **class Datasource_Type** 159 ⟩ ≡
  class Datasource_Type;
  typedef Datasource_Type ∗**Datasource_Node**;
  class Datasource_Type {
    ⟨ **friend** declarations for **class Datasource_Type** 160 ⟩
  private: **unsigned short** *type*;
    static map⟨**unsigned short**, **string**⟩ *datasource_type_map*;
    string *name*;
    void ∗*value*;
    Scanner_Node *scanner_node*;
  public: ⟨ Declare **static Datasource_Type** constants 161 ⟩
    ⟨ Declare **Datasource_Type** functions 167 ⟩
  };

This code is used in sections 182 and 183.

160. friend declarations for class datasource_type. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this section.

⟨ **friend** declarations for **class Datasource_Type** 160 ⟩ ≡
  friend int *yyparse*(**void** ∗);

This code is used in section 159.

161. Declare static datasource_type constants. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this section with the declaration of `DATASOURCE_TYPE_NULL_TYPE`.

[LDF 2006.12.08.] Added the declarations of `DBT_TYPE`, `GBV_GVK_TYPE`, `TIMMS_TYPE`, and `DATASOURCE_FILE_TYPE`.

⟨ Declare **static Datasource_Type** constants 161 ⟩ ≡
  static const unsigned short `DATASOURCE_TYPE_NULL_TYPE`;
  static const unsigned short `DBT_TYPE`;
  static const unsigned short `GBV_GVK_TYPE`;
  static const unsigned short `TIMMS_TYPE`;
  static const unsigned short `DATASOURCE_FILE_TYPE`;

See also section 164.

This code is used in section 159.

162.

⟨ Initialize **static Datasource_Type** constants 162 ⟩ ≡
const unsigned short Datasource_Type :: `DATASOURCE_TYPE_NULL_TYPE` = 0;
const unsigned short Datasource_Type :: `DBT_TYPE` = 1;
const unsigned short Datasource_Type :: `GBV_GVK_TYPE` = 2;
const unsigned short Datasource_Type :: `TIMMS_TYPE` = 3;
const unsigned short Datasource_Type :: `DATASOURCE_FILE_TYPE` = 4;

This code is used in section 182.

163. Initialize static Datasource_Type variables. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this section with the declaration of *datasource_type_map*.

⟨ Initialize **static Datasource_Type** variables 163 ⟩ ≡
map⟨**unsigned short**, **string**⟩ **Datasource_Type** :: *datasource_type_map*;

This code is used in section 182.

164. Types for *datasource_type*. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this section.

[LDF 2006.11.13.] Added **static const unsigned short** `TOP_TYPE`.

⟨ Declare **static Datasource_Type** constants 161 ⟩ +≡

165. datasource_type functions. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this section.

166. Constructor and setting functions. [LDF 2006.12.08.]

167. Default constructor. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this function.

⟨ Declare **Datasource_Type** functions 167 ⟩ ≡
Datasource_Type(**void**);

See also sections 169, 171, 175, 177, and 179.

This code is used in section 159.

168.

⟨ Define **Datasource_Type** functions 168 ⟩ ≡

```
Datasource_Type::Datasource_Type(void)
{
  cerr << "Entering␣‘Datasource_Type::Datasource_Type(void)’." << endl;
  return;
}   /* End of default Datasource_Type constructor definition. */
```

See also sections 170, 172, 173, 174, 176, 178, and 180.

This code is used in section 182.

169. Set. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this function.

⟨ Declare **Datasource_Type** functions 167 ⟩ +≡

```
int set(unsigned short datasource_type, Scanner_Node sscanner_node = 0);
```

170.

⟨ Define **Datasource_Type** functions 168 ⟩ +≡

```
int Datasource_Type::set(unsigned short datasource_type, Scanner_Node sscanner_node)
{
  return 0;
}   /* End of Datasource_Type::set definition. */
```

171. Destructor. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this function.

⟨ Declare **Datasource_Type** functions 167 ⟩ +≡

```
~Datasource_Type(void);
```

172.

⟨ Define **Datasource_Type** functions 168 ⟩ +≡

```
Datasource_Type::~Datasource_Type(void){
#if 0   /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣‘Datasource_Type’␣destructor." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    temp_strm.str("");
#endif
```

173.

⟨ Define **Datasource_Type** functions 168 ⟩ +≡

```
#ifdef DEBUG_OUTPUT
  temp_strm << "Exiting␣`Datasource_Type'␣destructor." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  temp_strm.str("");
#endif
```

174. DO NOT delete *scanner_node*! [LDF 2006.12.08.]

⟨ Define **Datasource_Type** functions 168 ⟩ +≡

```
  scanner_node = 0;
  return; }    /* End of ~Datasource_Type definition. */
```

175. Assignment. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this function.

⟨ Declare **Datasource_Type** functions 167 ⟩ +≡

```
  Datasource_Node operator=(const Datasource_Type &d);
```

176.

```
⟨ Define Datasource_Type functions 168 ⟩ +≡
  Datasource_Node Datasource_Type::operator=(const Datasource_Type &d)
  {
#if 0     /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
    scanner_node = d.scanner_node;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣‘Datasource_Type::operator=(const␣Datasource_Type)’." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (scanner_node) scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    if (this ≡ &d) {
#ifdef DEBUG_OUTPUT
      temp_strm << "In␣‘Datasource_Type::operator=(const␣Datasource_Type)’:" << endl <<
          "Self-assignment.␣␣Returning␣‘this’." << endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      if (scanner_node) scanner_node->log_strm << temp_strm.str();
      temp_strm.str("");
#endif
      return this;
    }    /* if (this ≡ &d) */
    type = d.type;
#ifdef DEBUG_OUTPUT
    temp_strm << "Exiting␣‘Datasource_Type::operator=(const␣Datasource_Type)’." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (scanner_node) scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    return this;
  }    /* End of Datasource_Type::operator=(const Datasource_Type d) definition. */
```

177. Initialize type maps. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this function.

[LDF 2006.12.08.] Added code for `DBT_TYPE`, `GBV_GVK_TYPE`, `TIMMS_TYPE`, and `DATASOURCE_FILE_TYPE`.

⟨ Declare **Datasource_Type** functions 167 ⟩ +≡

```
static int initialize_type_maps(void);
```

178.

⟨ Define **Datasource_Type** functions 168 ⟩ +≡

```
int Datasource_Type::initialize_type_maps(void)
{
  datasource_type_map[DATASOURCE_TYPE_NULL_TYPE] = "DATASOURCE_TYPE_NULL_TYPE";
  datasource_type_map[DBT_TYPE] = "DBT_TYPE";
  datasource_type_map[GBV_GVK_TYPE] = "GBV_GVK_TYPE";
  datasource_type_map[TIMMS_TYPE] = "TIMMS_TYPE";
  datasource_type_map[DATASOURCE_FILE_TYPE] = "DATASOURCE_FILE_TYPE";
  return 0;
}   /* End of Datasource_Type::initialize_type_maps definition. */
```

179. Show. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this function.

⟨ Declare **Datasource_Type** functions 167 ⟩ +≡

```
int show(string s = "`Datasource_Type':\n", Scanner_Node sscanner_node = 0);
```

180.

⟨ Define **Datasource_Type** functions 168 ⟩ +≡

```
int Datasource_Type::show(string s, Scanner_Node sscanner_node)
{
  stringstream temp_strm;
  if (scanner_node ≡ 0 ∧ sscanner_node ≠ 0) scanner_node = sscanner_node;
  temp_strm ≪ s ≪ endl ≪ "`type'␣==␣" ≪ datasource_type_map[type] ≪ "␣(" ≪ type ≪ ")" ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  temp_strm.str("");
  if (scanner_node ≠ 0 ∧ scanner_node→log_strm.is_open()) scanner_node→log_strm ≪ temp_strm;
  return 0;
}   /* End of Datasource_Type::show definition. */
```

181. Putting datasource_type together. [LDF 2006.12.08.]

182. This is what's compiled.

```
⟨ Include files 10 ⟩
⟨ dtsrctyp.web 155 ⟩
using namespace std;
using namespace Scan_Parse;
⟨ Forward declarations 13 ⟩
⟨ Declare class Datasource_Type 159 ⟩
⟨ Initialize static Datasource_Type constants 162 ⟩
⟨ Initialize static Datasource_Type variables 163 ⟩
⟨ Define Datasource_Type functions 168 ⟩
```

183. This is what's written to `dtsrctyp.h`.

```
⟨ dtsrctyp.hh 183 ⟩ ≡
#ifndef DTSRCTYP_KNOWN
#define DTSRCTYP_KNOWN 1
  ⟨ Preprocessor macro calls 19 ⟩
  using namespace std;
  ⟨ Forward declarations 13 ⟩
  ⟨ Declare class Datasource_Type 159 ⟩
#endif  /* DTSRCTYP_KNOWN is defined. */
```

184. id_type (`idtype.web`). [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Created this file.

```
⟨ idtype.web 184 ⟩ ≡
  static char id_string[] = "$Id:␣idtype.web,v␣1.4␣2007/01/02␣12:19:12␣lfinsto1␣Exp␣$";
```

This code is cited in sections 6 and 8.

This code is used in section 207.

185. Include files.

```
⟨ Include files 10 ⟩ +≡
#include "localldf.h"
#ifdef NON_WIN_LDF
#include "nonwin.h"
#else
#include "stdafx.h"
#endif
#include "dttmtype.h"
#include "parser.hxx"
#include "scnrtype.h"
#include "querytyp.h"
```

186. Preprocessor macro calls.

```
⟨ Preprocessor macro calls 19 ⟩ +≡
#ifdef WIN_LDF
#pragma once
#endif
```

187. Forward declarations. [LDF 2006.10.31.]

⟨ Forward declarations 13 ⟩ +≡
  class Scanner_Type;
  typedef Scanner_Type ∗**Scanner_Node**;
  struct Id_Type;
  typedef Id_Type ∗**Id_Node**;

188. struct id_type declaration. [LDF 2006.10.17.]

Log

[LDF 2006.11.21.] Added **unsigned short** *subtype*. It's needed for *strings*.

[LDF 2006.12.01.] Added **static map⟨unsigned short, string⟩** *subtype_map*.

⟨ Declare **struct Id_Type** 188 ⟩ ≡
  struct Id_Type {
    friend class Scanner_Type;
    string *name*;
    void ∗*value*;
    unsigned short *type*;
    unsigned short *subtype*;
    Id_Node *up*;
    Id_Node *left*;
    Id_Node *right*;
    static map⟨unsigned short, string⟩ *subtype_map*;
    Scanner_Node *scanner_node*;
    ⟨ Declare **Id_Type** functions 195 ⟩
    ⟨ Declare **static Id_Type** constants 189 ⟩
  };

This code is used in sections 207 and 208.

189. static Id_Type constants.

Log

[2006.11.21.] Added this section with the declarations of the **static const unsigned** *shorts* `TEX_STRING_TYPE`, `SQL_STRING_TYPE`, and `PQF_STRING_TYPE`. These types must be declared in **struct Id_Type**, because the Standard C++ library type **string** is used to represent the **string** type in the Scantest language.

⟨ Declare **static Id_Type** constants 189 ⟩ ≡
  static const unsigned short `TEX_STRING_TYPE`;
  static const unsigned short `SQL_STRING_TYPE`;
  static const unsigned short `PQF_STRING_TYPE`;

This code is used in section 188.

190.

Log

[2006.11.21.] Added this section with the initializations of Added this section with the declarations of the **static const unsigned** *shorts* `TEX_STRING_TYPE`, `SQL_STRING_TYPE`, and `PQF_STRING_TYPE`.

⟨ Initialize **static Id_Type** constants 190 ⟩ ≡
  const unsigned short Id_Type ::`TEX_STRING_TYPE` = 1;
  const unsigned short Id_Type ::`SQL_STRING_TYPE` = 2;
  const unsigned short Id_Type ::`PQF_STRING_TYPE` = 3;

This code is used in section 207.

191. Declare **static Id_Type** variables. [LDF 2006.12.01.]

Id_Type :: *subtype_map* needs to be declared outside of the class declaration. It is initialized in the function *initialize_subtype_map*, which is called in *main*. [LDF 2006.12.01.]

Log

[2006.12.01.] Added this section.

⟨ Declare **static Id_Type** variables 191 ⟩ ≡
  map⟨**unsigned short**, **string**⟩ **Id_Type** :: *subtype_map*;

This code is used in section 207.

192. id_type functions. [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Added this section.

---

193. Constructors and setting functions. [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Added this section.

---

194. Default constructor. [LDF 2006.11.01.]

Log

[2006.10.31.] Added this function.

[LDF 2006.11.02.] Now setting **Scanner_Node** *scanner_node* = 0.

195.

⟨ Declare **Id_Type** functions 195 ⟩ ≡
  Id_Type(**void**);

See also sections 198, 201, and 204.

This code is used in section 188.

196.

⟨ Define **Id_Type** functions 196 ⟩ ≡

```
Id_Type::Id_Type(void)
{
   name = "";
   value = 0;
   type = 0;
   subtype = 0;
   up = 0;
   left = 0;
   right = 0;
   scanner_node = 0;
   return;
}    /* End of the default Id_Type constructor definition. */
```

See also sections 199, 202, and 205.
This code is used in section 207.

197. Destructor. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this function.

198.

⟨ Declare **Id_Type** functions 195 ⟩ +≡

```
~Id_Type(void);
```

199.

```
⟨ Define Id_Type functions 196 ⟩ +≡
  Id_Type :: ~Id_Type(void)
  {
#if 0     /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "Entering␣`~Id_Type'." ≪ endl ≪ "`name'␣==␣`" ≪ name ≪ "'" ≪ endl ≪
        "`type'␣==␣`" ≪ Scan_Parse :: token_map[type] ≪ "'" ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    if (type ≡ QUERY_TYPE) {
      delete static_cast⟨Query_Node⟩(value);
      value = 0;
    }
    delete left;
    delete right;
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "Exiting␣`~Id_Type'." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    return;
#undef DEBUG_OUTPUT
  }     /* End of ~Id_Type definition. */
```

200. Initialize *subtype_map*. [LDF 2006.12.01.]

Log

[LDF 2006.12.01.] Added this function.

201.

```
⟨ Declare Id_Type functions 195 ⟩ +≡
  static int initialize_subtype_map(Scanner_Node s = 0);
```

202.

⟨ Define **Id_Type** functions 196 ⟩ +≡

```
int Id_Type::initialize_subtype_map(Scanner_Node s)
{
#if 0    /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣`Id_Type::initialize_subtype_map'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (s ≠ 0) s->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Id_Type::subtype_map[TEX_STRING_TYPE] = "TEX_STRING_TYPE";
    Id_Type::subtype_map[SQL_STRING_TYPE] = "SQL_STRING_TYPE";
    Id_Type::subtype_map[PQF_STRING_TYPE] = "PQF_STRING_TYPE";
#ifdef DEBUG_OUTPUT
    temp_strm << "Exiting␣`Id_Type::initialize_subtype_map'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (s ≠ 0) s->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    return 0;
#undef DEBUG_OUTPUT
}   /* End of Id_Type::initialize_subtype_map definition. */
```

203. Show. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this function.

204.

⟨ Declare **Id_Type** functions 195 ⟩ +≡

```
int show(string = "");
```

205.

⟨ Define **Id_Type** functions 196 ⟩ +≡

```
  int Id_Type::show(string s)
  {
#if 0    /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
    if (s ≡ "") s = "`Id_Type:'";
    temp_strm ≪ s ≪ endl ≪ "`name'␣==␣`" ≪ name ≪ "'" ≪ endl ≪ "`type'␣==␣`" ≪
        Scan_Parse::token_map[type] ≪ "'" ≪ endl ≪ "`subtype'␣==␣" ≪ subtype_map[subtype] ≪
        endl;
    if (value ≡ 0) temp_strm ≪ "`value'␣␣==␣0" ≪ endl;
    else temp_strm ≪ "`value'␣␣!=␣0" ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "Exiting␣`Id_Type::show'." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    return 0;
#undef DEBUG_OUTPUT
  }    /* End of Id_Type::show definition. */
```

206. Putting id_type together. [LDF 2006.11.01.]

207. This is what's compiled.

```
  ⟨ Include files 10 ⟩
  ⟨ idtype.web 184 ⟩
  using namespace std;
  ⟨ Forward declarations 13 ⟩
  ⟨ Declare struct Id_Type 188 ⟩
  ⟨ Initialize static Id_Type constants 190 ⟩
  ⟨ Declare static Id_Type variables 191 ⟩
  ⟨ Define Id_Type functions 196 ⟩
```

208. This is what's written to `idtype.h`.

⟨ `idtype.hh` 208 ⟩ ≡
  ⟨ Preprocessor macro calls 19 ⟩
  using namespace std;
  ⟨ Forward declarations 13 ⟩
  ⟨ Declare **struct Id_Type** 188 ⟩

209. Scanner_Type (`scnrtype.web`). [LDF 2006.10.17.]

Log

[LDF 2006.10.31.] Created this file.

⟨ `scnrtype.web` 209 ⟩ ≡
  static char *id_string*[ ] = "$Id:␣scnrtype.web,v␣1.4␣2007/01/02␣12:19:56␣lfinsto1␣Exp␣$";

This code is cited in sections 6 and 8.

This code is used in section 243.

210. Include files.

⟨ Include files 10 ⟩ +≡
```
#include "localldf.h"
#ifdef NON_WIN_LDF
#include "nonwin.h"
#else
#include "stdafx.h"
#endif
#include "parser.hxx"
#include "dttmtype.h"
#include "idtype.h"
#include "scanner.h"
#include "querytyp.h"
#include "dtsrctyp.h"
```

211. Preprocessor macro calls.

⟨ Preprocessor macro calls 19 ⟩ +≡
```
#ifdef WIN_LDF
#pragma once
#endif
```

212. Forward declarations. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this section.

⟨ Forward declarations 13 ⟩ +≡
  class Scanner_Type;
  typedef Scanner_Type ∗**Scanner_Node**;
  class Query_Type;
  typedef Query_Type ∗**Query_Node**;
  class Id_Type;

typedef Id_Type $*$**Id_Node**;

213. struct Token_Type declaration. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this declaration.

[LDF 2006.11.02.] Added constructor.

⟨ Declare **struct Token_Type** 213 ⟩ ≡
```
struct Token_Type {
   unsigned int type;
   YYSTYPE value;
   ⟨ Declare Token_Type functions 215 ⟩
};
```
This code is used in sections 243 and 244.

214. Token_Type functions. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this section.

---

215. Default Constructor. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this function.

⟨ Declare **Token_Type** functions 215 ⟩ ≡
```
Token_Type(void);
```
See also section 217.

This code is used in section 213.

216.

⟨ Define **Token_Type** functions 216 ⟩ ≡
```
Token_Type::Token_Type(void)
{
   type = 0;
   value.int_value = 0;
   return;
}
```
See also section 218.

This code is used in section 243.

217. Non-Default Constructor. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this function.

⟨ Declare **Token_Type** functions 215 ⟩ +≡
Token_Type(**unsigned int** *ttype*, YYSTYPE *vvalue*);

218.

⟨ Define **Token_Type** functions 216 ⟩ +≡
Token_Type :: **Token_Type**(**unsigned int** *ttype*, YYSTYPE *vvalue*): *type*(*ttype*), *value*(*vvalue*)
{
return;
}

219. class Scanner_Type declaration. This is the type passed to the functions *yylex* and *yyparse* as a parameter. [LDF 2006.10.17.]

Log

[LDF 2006.10.31.] Added **typedef Scanner_Type** ∗**Scanner_Node**.

[LDF 2006.10.31.] Working on this declaration.

[LDF 2006.11.02.] Added **Query_Node** *curr_query*.

[LDF 2006.11.03.] Added **vector**⟨**float**⟩ *float_vector*.

⟨ Declare **class Scanner_Type** 219 ⟩ ≡
class Scanner_Type { ⟨ **friend** declarations for **Scanner_Type** 220 ⟩
char *in_filename*[128];
char *log_filename*[128];
ifstream *in_strm*;
map⟨**string**, **Id_Type** ∗⟩ *id_map*; *stack* < **Token_Type** > *token_stack*;
vector⟨**float**⟩ *float_vector*;
public: **ofstream** *log_strm*;
⟨ Declare **Scanner_Type** functions 226 ⟩
} ;

This code is used in sections 243 and 244.

220. friend declarations for Scanner_Type. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

⟨ **friend** declarations for **Scanner_Type** 220 ⟩ ≡
friend int *main*(**int**, **char** ∗[ ]); **friend int** *yylex* (YYSTYPE ∗ , **void** ∗) ;
friend int *yyparse*(**void** ∗);

See also section 222.

This code is used in section 219.

221. friend declarations for parser rule functions. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

---

222. friend declarations for functions for group statements. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section with the **friend** declarations for **Scan_Parse** :: *start_local_query_func* and **Scan_Parse** :: *end_local_query_func*.

[LDF 2006.11.01.] Changed the name of **Scan_Parse** :: *start_local_query_func* to **Scan_Parse** :: *start_local_database_query_func* and the name of **Scan_Parse** :: *end_local_query_func* to **Scan_Parse** :: *end_query_func*.

[LDF 2006.11.01.] Added the **friend** declaration of **Scan_Parse** :: *declare_variable_func*.

[LDF 2006.11.01.] Added the **char** ∗*name* argument to **Scan_Parse** :: *declare_variable_func*.

[LDF 2006.11.02.] Changed the return value of **Scan_Parse** :: *declare_variable_func* from **int** to **void** ∗.

[LDF 2006.11.02.] Added the **friend** declaration of **Scan_Parse** :: *variable_func*.

[LDF 2006.11.02.] Added the **friend** declaration of **Scan_Parse** :: *query_assignment_func_0*.

[LDF 2006.12.15.] Added the **friend** declaration of **Scan_Parse** :: *datetime_assignment_func_0*.

[LDF 2006.12.18.] Added the **friend** declaration of **Scan_Parse** :: *datetime_assignment_func_1*.

[LDF 2006.12.19.] Added the **friend** declaration of **Scan_Parse** :: *query_assignment_func_1*.

⟨ **friend** declarations for **Scanner_Type** 220 ⟩ +≡
friend void ∗**Scan_Parse** :: *variable_func*(**void** ∗*v*, **char** ∗*name*);
friend Id_Node Scan_Parse :: *declare_variable_func*(**void** ∗*v*, **const unsigned short** *type*, **char** ∗*name*);
friend void ∗**Scan_Parse** :: *query_assignment_func_0*(**void** ∗*v*, **void** ∗*object*, **unsigned int** *assignment_type*, **unsigned int** *arg_0*, **unsigned int** *arg_1*, **void** ∗*value*, **bool** *negate*);
friend int Scan_Parse :: *query_assignment_func_1*(**Scanner_Node** *scanner_node*, **Id_Node** &*curr_id_node*, **void** ∗&*field_specifier*, **int** *assignment_operator*, **int** *negation_optional*, **int** *match_term_optional*, **void** ∗&*v*, **int** *type*);
friend int Scan_Parse :: *start_local_database_query_func*(**void** ∗*v*);
friend int Scan_Parse :: *end_query_func*(**void** ∗*v*);
friend int Scan_Parse :: *datetime_assignment_func_0*(**void** ∗*v*, **Date_Time_Node** &*curr_date_time_node*, **int** *specifier*, **int** *op*, **void** ∗*val*, **int** *type*);
friend int Scan_Parse :: *datetime_assignment_func_1*(**void** ∗*v*, **Date_Time_Node** &*d*, **int** *op*, **void** ∗&*vec*);

223. Scanner_Type functions. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

---

224. Constructors and setting functions. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

---

225. Default constructor. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this function.

[LDF 2006.11.02.] Now initializing **Query_Node** *query_node* to 0.

226.

⟨ Declare **Scanner_Type** functions 226 ⟩ ≡
```
Scanner_Type(void);
```
See also sections 228, 230, and 232.

This code is used in section 219.

227.

⟨ Define **Scanner_Type** functions 227 ⟩ ≡
```
Scanner_Type::Scanner_Type(void)
{
  return;
}   /* End of the default Scanner_Type constructor definition. */
```
See also sections 229, 231, 233, 234, 235, 236, 237, 238, 239, 240, and 241.

This code is used in section 243.

228. Destructor. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this function.

⟨ Declare **Scanner_Type** functions 226 ⟩ +≡
```
~Scanner_Type(void);
```

229.

```
⟨ Define Scanner_Type functions 227 ⟩ +≡
  Scanner_Type::~Scanner_Type(void)
  {
#if 0    /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣`~Scanner_Node'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (log_strm.is_open()) log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    for (map<string, Id_Type *>::iterator iter = id_map.begin(); iter ≠ id_map.end(); ++iter) {
      delete iter->second;
    }
    id_map.clear();
    in_strm.close();
#ifdef DEBUG_OUTPUT
    temp_strm << "Exiting␣`~Scanner_Node'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (log_strm.is_open()) log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    log_strm.close();
    return;
#undef DEBUG_OUTPUT
  }    /* End of the ~Scanner_Type definition. */
```

230. Initialize *id_map*. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this function.

[LDF 2006.11.14.] Removed the code that declared a variable "curr_query" and set *this→curr_query* to point to it. I now plan to use *curr_query* for parsing subqueries.

```
⟨ Declare Scanner_Type functions 226 ⟩ +≡
  int initialize_id_map(void);
```

231.

```
⟨ Define Scanner_Type functions 227 ⟩ +≡
  int Scanner_Type::initialize_id_map(void)
  {
#if 0   /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    stringstream temp_strm;
#ifdef DEBUG_OUTPUT
    temp_strm << "Entering␣‘Scanner_Node::initialize_id_map’." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (log_strm.is_open()) log_strm << temp_strm.str();
    temp_strm.str("");
#endif
#ifdef DEBUG_OUTPUT
    temp_strm << "Exiting␣‘Scanner_Node::initialize_id_map’." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (log_strm.is_open()) log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    return 0;
#undef DEBUG_OUTPUT
  }   /* End of the Scanner_Type::initialize_id_map definition. */
```

232. Lookup. [LDF 2006.11.03.]

Log

[LDF 2006.11.03.] Added this function.

```
⟨ Declare Scanner_Type functions 226 ⟩ +≡
  Id_Node lookup(char *name, bool create = false, unsigned int type = NULL_TYPE);
```

233.

```
⟨ Define Scanner_Type functions 227 ⟩ +≡
  Id_Node Scanner_Type::lookup(char *name, bool create, unsigned int type){
#if 0      /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
      stringstream temp_strm;
#ifdef DEBUG_OUTPUT
      temp_strm ≪ "Entering␣‘Scanner_Node::lookup’." ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      Id_Node curr_id_node;
      map⟨string, Id_Node⟩::iterator iter = id_map.find(name);
      if (iter ≡ id_map.end()) {
        curr_id_node = 0;
      }
      else      /* iter ≠ id_map.end() */
      { curr_id_node = iter→second;
```

234. WARNING: *type* ≠ *curr_id_node→type*. [LDF 2006.11.03.]

```
⟨ Define Scanner_Type functions 227 ⟩ +≡
  if (type ≠ curr_id_node→type) {
    temp_strm ≪ "WARNING!␣␣In␣‘Scanner_Node::lookup’:" ≪ endl ≪
        "‘type’␣==␣‘" ≪ Scan_Parse::token_map[type] ≪ "’␣is␣not␣the␣same␣as␣" ≪
        "‘curr_id_node->type’␣==␣‘" ≪ Scan_Parse::token_map[curr_id_node→type] ≪ "’." ≪
        endl ≪ "Setting␣‘type’␣to␣‘" ≪ Scan_Parse::token_map[curr_id_node→type] ≪
        "’␣and␣continuing." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    log_strm ≪ temp_strm.str();
    temp_strm.str("");
    type = curr_id_node→type;
  }      /* if (type ≠ curr_id_node→type) */
```

235.

```
⟨ Define Scanner_Type functions 227 ⟩ +≡
  }      /* else (iter ≠ id_map.end()) */
```

236. *name* doesn't refer to an array. [LDF 2006.11.03.]

```
⟨ Define Scanner_Type functions 227 ⟩ +≡
  if (strchr(name, '¥') ≡ 0) {
    if (curr_id_node ≠ 0) return curr_id_node;
    else if (create ≡ true) {
      curr_id_node = new Id_Type;
      curr_id_node→name = name;
      curr_id_node→type = type;
      id_map[name] = curr_id_node;
      return curr_id_node;
    }
    else      /* curr_id_node ≡ 0 and create ≡ false */
    {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ "In␣`Scanner_Node::lookup':" ≪ "`name'␣not␣found␣in␣`id_map'␣and␣`creat\
          e'␣==␣0." ≪ endl ≪ "Exiting␣function␣with␣return␣value␣0.";
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      return 0;
    }      /* else (curr_id_node ≡ 0 and create ≡ false) */
  }      /* if (strchar(name, '¥') ≡ 0) */
```

237. *name* refers to an array. [LDF 2006.11.03.]

```
⟨ Define Scanner_Type functions 227 ⟩ +≡
  else      /* strchar(name, '¥') ≠ 0 */
  { stringstream curr_name_strm;
  string curr_name;
  {      /* Beginning of group */
    int j = 0;
    for (int i = 0; i < strlen(name); ++i) {
      if (name[i] ≡ '¥') curr_name_strm ≪ '[' ≪ float_vector[j++] ≪ ']';
      else curr_name_strm ≪ name[i];
    }      /* for */
  }      /* End of group */
  curr_name = curr_name_strm.str();
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "`curr_name'␣==␣" ≪ curr_name ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

238. No elements on array. [LDF 2006.11.03.]

⟨ Define **Scanner_Type** functions 227 ⟩ +≡

```
  if (curr_id_node→left ≡ 0) {
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "‘" ≪ name ≪ "’␣refers␣to␣an␣empty␣array." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

239. *create* ≡ *false*. Return 0. [LDF 2006.11.03.]

⟨ Define **Scanner_Type** functions 227 ⟩ +≡

```
  if (create ≡ false) {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "‘create’␣==␣‘false’.␣␣Returning␣0." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    return 0;
  }    /* if (create ≡ false) */
```

240. *create* ≡ *true*. Create array element. [LDF 2006.11.03.]

⟨ Define **Scanner_Type** functions 227 ⟩ +≡

```
  else    /* create ≡ true */
  {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "Creating␣array␣element␣for␣‘" ≪ curr_name ≪ "’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    curr_id_node→left = new Id_Type;
    curr_id_node→left→name = curr_name;
    curr_id_node→left→type = type;
    curr_id_node→left→up = curr_id_node;
    return curr_id_node→left;
  }    /* else (create ≡ true) */
  }    /* if (curr_id_node→left ≡ 0) */
```

241. Array isn't empty. Traverse the tree. [LDF 2006.11.03.]

⟨ Define **Scanner_Type** functions 227 ⟩ +≡

```
  else    /* curr_id_node→left ≠ 0 */
  {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "`" ≪ name ≪ "'␣array␣isn't␣empty.␣␣" ≪ "Traversing." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    curr_id_node = curr_id_node→left;
    for ( ; ; ) {
      if (curr_name ≡ curr_id_node→name) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "`" ≪ curr_name ≪ "'␣exists.␣␣Returning␣it." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return curr_id_node;
      }    /* if (curr_name ≡ curr_id_node→name) */
      else if (curr_name.compare(curr_id_node→name) < 0) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "`" ≪ curr_name ≪ "'␣<␣`" ≪ curr_id_node→name ≪ "'.␣␣" ≪
            "Traversing␣the␣tree␣leftwards." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        if (curr_id_node→left ≠ 0) {
#ifdef DEBUG_OUTPUT
          temp_strm ≪ "`" ≪ curr_id_node→name ≪ "->left'␣exists.␣␣" ≪ "Continuing." ≪ endl;
          cerr_mutex.Lock();
          cerr ≪ temp_strm.str();
          cerr_mutex.Unlock();
          log_strm ≪ temp_strm.str();
          temp_strm.str("");
#endif
          curr_id_node = curr_id_node→left;
          continue;
        }
        else    /* curr_id_node→left ≡ 0 */
        {
#ifdef DEBUG_OUTPUT
          temp_strm ≪ "`" ≪ curr_id_node→name ≪ "->left'␣doesn't␣exist." ≪ endl;
          cerr_mutex.Lock();
```

```
            cerr ≪ temp_strm.str();
            cerr_mutex.Unlock();
            log_strm ≪ temp_strm.str();
            temp_strm.str("");
#endif
            if (create ≡ true) {
#ifdef DEBUG_OUTPUT
              temp_strm ≪ "Creating␣new␣‘Id_Node’␣for␣‘" ≪ curr_name ≪ "’." ≪ endl;
              cerr_mutex.Lock();
              cerr ≪ temp_strm.str();
              cerr_mutex.Unlock();
              log_strm ≪ temp_strm.str();
              temp_strm.str("");
#endif
              curr_id_node→left = new Id_Type;
              curr_id_node→left→name = curr_name;
              curr_id_node→left→type = type;
              curr_id_node→left→up = curr_id_node;
              return curr_id_node→left;
            }   /* if (create ≡ true) */
            else   /* create ≡ false */
            {
#ifdef DEBUG_OUTPUT
              temp_strm ≪ "‘create’␣==␣‘false’.␣␣" ≪ "Not␣creating␣new␣‘Id_Node’␣for␣‘" ≪
                  curr_name ≪ "’.␣␣Returning␣0." ≪ endl;
              cerr_mutex.Lock();
              cerr ≪ temp_strm.str();
              cerr_mutex.Unlock();
              log_strm ≪ temp_strm.str();
              temp_strm.str("");
#endif
              return 0;
            }   /* else (create ≡ false) */
          }   /* else (curr_id_node→left ≡ 0) */
        }   /* else if (curr_name.compare(curr_id_node→name) < 0) */
        else if (curr_name.compare(curr_id_node→name) > 0) {
#ifdef DEBUG_OUTPUT
          temp_strm ≪ "‘" ≪ curr_name ≪ "’␣>␣‘" ≪ curr_id_node→name ≪ "’.␣␣" ≪
              "Traversing␣the␣tree␣rightwards." ≪ endl;
          cerr_mutex.Lock();
          cerr ≪ temp_strm.str();
          cerr_mutex.Unlock();
          log_strm ≪ temp_strm.str();
          temp_strm.str("");
#endif
          if (curr_id_node→right ≠ 0) {
#ifdef DEBUG_OUTPUT
            temp_strm ≪ "‘" ≪ curr_id_node→name ≪ "->right’␣exists.␣␣" ≪ "Continuing." ≪ endl;
            cerr_mutex.Lock();
            cerr ≪ temp_strm.str();
            cerr_mutex.Unlock();
            log_strm ≪ temp_strm.str();
```

```
            temp_strm.str("");
#endif
            curr_id_node = curr_id_node→right;
            continue;
          }   /* if (curr_id_node→right ≠ 0) */
          else   /* curr_id_node→right ≡ 0 */
          {
#ifdef DEBUG_OUTPUT
            temp_strm ≪ "`" ≪ curr_id_node→name ≪ "->right'␣doesn't␣exist." ≪ endl;
            cerr_mutex.Lock();
            cerr ≪ temp_strm.str();
            cerr_mutex.Unlock();
            log_strm ≪ temp_strm.str();
            temp_strm.str("");
#endif
            if (create ≡ true) {
#ifdef DEBUG_OUTPUT
              temp_strm ≪ "Creating␣new␣`Id_Node'␣for␣`" ≪ curr_name ≪ "'." ≪ endl;
              cerr_mutex.Lock();
              cerr ≪ temp_strm.str();
              cerr_mutex.Unlock();
              log_strm ≪ temp_strm.str();
              temp_strm.str("");
#endif
              curr_id_node→right = new Id_Type;
              curr_id_node→right→name = curr_name;
              curr_id_node→right→type = type;
              curr_id_node→right→up = curr_id_node;
              return curr_id_node→right;
            }   /* if (create ≡ true) */
            else   /* create ≡ false */
            {
#ifdef DEBUG_OUTPUT
              temp_strm ≪ "`create'␣==␣`false'.␣␣" ≪ "Not␣creating␣new␣`Id_Node'␣for␣`" ≪
                  curr_name ≪ "'.␣␣Returning␣0." ≪ endl;
              cerr_mutex.Lock();
              cerr ≪ temp_strm.str();
              cerr_mutex.Unlock();
              log_strm ≪ temp_strm.str();
              temp_strm.str("");
#endif
              return 0;
            }   /* else (create ≡ false) */
          }   /* else (curr_id_node→right ≡ 0) */
        }   /* else if (curr_name.compare(curr_id_node→name) > 0) */
      }   /* for */
    }   /* else (curr_id_node→left ≠ 0) */
  }   /* else (strchar(name, '¥') ≠ 0) */
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "Exiting␣`Scanner_Node::lookup'." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
```

```
  cerr_mutex.Unlock( );
  log_strm ≪ temp_strm.str( );
  temp_strm.str("");
#endif
  return 0;
#undef DEBUG_OUTPUT
  }    /* End of Scanner_Type::lookup definition. */
```

242. Putting Scanner_Type together. [LDF 2006.10.31.]

243. This is what's compiled.

```
  ⟨ Include files 10 ⟩
  ⟨ scnrtype.web 209 ⟩
  using namespace std;
  ⟨ Forward declarations 13 ⟩
  ⟨ Declare struct Token_Type 213 ⟩
  ⟨ Declare class Scanner_Type 219 ⟩
  ⟨ Define Token_Type functions 216 ⟩
  ⟨ Define Scanner_Type functions 227 ⟩
```

244. This is what's written to `scnrtype.h`.

```
⟨ scnrtype.hh 244 ⟩ ≡
#ifndef SCNRTYPE_KNOWN
#define SCNRTYPE_KNOWN 1
  ⟨ Preprocessor macro calls 19 ⟩
  using namespace std;
#include "scanner.h"
  using namespace Scan_Parse;
  ⟨ Forward declarations 13 ⟩
  ⟨ Declare struct Token_Type 213 ⟩
  ⟨ Declare class Scanner_Type 219 ⟩
#endif   /* SCNRTYPE_KNOWN is defined. */
```

245. Scanning (`scanner.web`). [LDF 2006.10.17.]

Log

[LDF 2006.10.17.] Created this file.

```
⟨ scanner.web 245 ⟩ ≡
  static char id_string[] = "$Id:␣scanner.web,v␣1.4␣2007/01/02␣12:21:05␣lfinsto1␣Exp␣$";
```

This code is cited in sections 6 and 8.

This code is used in section 357.

246. Include files.

```
⟨ Include files 10 ⟩ +≡
#include "localldf.h"
#ifdef NON_WIN_LDF
#include "nonwin.h"
#else
#include "stdafx.h"
#endif
#include "parser.hxx"
#include "dttmtype.h"
#include "idtype.h"
#include "scnrtype.h"
```

247. Preprocessor macro calls.

```
⟨ Preprocessor macro calls 19 ⟩ +≡
#ifdef WIN_LDF
#pragma once
#endif
```

248. Forward declarations. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this section.

```
⟨ Forward declarations 13 ⟩ +≡
  class Scanner_Type;
```

```
typedef Scanner_Type *Scanner_Node;
class Query_Type;
typedef Query_Type *Query_Node;
class Id_Type;
typedef Id_Type *Id_Node;
class Date_Time_Type;
typedef Date_Time_Type *Date_Time_Node;
```

249. Type declarations for the scanner. [LDF 2006.10.17.]

250. Declare namespace Scan_Parse. [LDF 2006.10.17.]

251. Declaration with initializations for compiliation. [LDF 2006.10.19.]

Log

[LDF 2006.10.19.] Added this section.

[LDF 2006.11.01.] Removed **const unsigned int** `COLLECTING_ARGUMENT`. Changed **const unsigned int** `COLLECTING_KEYWORD` to `COLLECTING_ID`.

[LDF 2006.11.02.] Added **extern const unsigned int** `COLLECTING_INTEGER` and **extern const unsigned int** `COLLECTING_FLOAT`.

⟨ Declare namespace **Scan_Parse** 251 ⟩ ≡

```
namespace Scan_Parse {
   extern const unsigned int NULL_STATE = 0;
   extern const unsigned int COLLECTING_ID = 1;
   extern const unsigned int COLLECTING_STRING = 2;
   extern const unsigned int COLLECTING_INTEGER = 3;
   extern const unsigned int COLLECTING_FLOAT = 4;
   ⟨ Declare struct Keyword_Type 257 ⟩
   ⟨ Declare keyword_map 259 ⟩
   ⟨ Declare token_map 262 ⟩
   ⟨ Declare Scan_Parse functions 264 ⟩
}   /* End of namespace Scan_Parse declaration. */
```

This code is used in section 357.

252. extern declaration of namespace Scan_Parse. [LDF 2006.10.19.]

⟨ **extern** declaration of namespace **Scan_Parse** 252 ⟩ ≡

```
namespace Scan_Parse {
```

See also sections 254 and 255.

This code is used in section 358.

253. Constants. [LDF 2006.10.19.]

254. Scanner state ("start conditions"). [LDF 2006.10.19.]

⟨ **extern** declaration of namespace **Scan_Parse** 252 ⟩ +≡
```
extern const unsigned int NULL_STATE;
extern const unsigned int COLLECTING_ID;
extern const unsigned int COLLECTING_STRING;
extern const unsigned int COLLECTING_INTEGER;
extern const unsigned int COLLECTING_FLOAT;
```

255.

⟨ **extern** declaration of namespace **Scan_Parse** 252 ⟩ +≡
```
struct Keyword_Type;
⟨ extern keyword_map declaration 260 ⟩
⟨ extern token_map declaration 263 ⟩
⟨ Declare Scan_Parse functions 264 ⟩
}   /* End of extern declaration of namespace Scan_Parse. */
```

256. Keywords. [LDF 2006.10.19.]

257. Declare struct Keyword_Type. [LDF Undated.]

Log

[LDF 2006.11.01.] Tried to change **Keyword_Type** from a **struct** to a **class**, but it didn't work.

⟨ Declare **struct Keyword_Type** 257 ⟩ ≡
```
struct Keyword_Type {
  string name;
  int value;
  string value_name;
};
```
This code is used in section 251.

258. Keyword map. [LDF 2006.10.19.]

Log

[LDF 2006.10.19.] Added this section.

---

259. Declare *keyword_map*. [LDF 2006.10.19.]

⟨ Declare *keyword_map* 259 ⟩ ≡
```
map⟨string, Keyword_Type *⟩ keyword_map;
```
This code is used in section 251.

260. extern declaration of *keyword_map* for the header file. [LDF 2006.10.19.]

⟨ **extern** *keyword_map* declaration 260 ⟩ ≡
  extern map⟨**string**, **Keyword_Type** ∗⟩ *keyword_map*;

This code is used in section 255.

261. Token map. [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Added this section.

---

262. Declare ***token_map*****.** [LDF 2006.11.01.]

⟨ Declare *token_map* 262 ⟩ ≡
  map⟨**unsigned int**, **string**⟩ *token_map*;

This code is used in section 251.

263. extern declaration of *token_map* for the header file. [LDF 2006.11.01.]

⟨ **extern** *token_map* declaration 263 ⟩ ≡
  extern map⟨**unsigned int**, **string**⟩ *token_map*;

This code is used in section 255.

264. Initialize ***keyword_map*** **and** ***token_map*****.** [LDF 2006.10.19.]

Log

[LDF 2006.11.03.] Added code for `NULL_TYPE`.

[LDF 2006.12.07.] Added code for `HYPHEN`.

[LDF 2006.12.12.] Added code for `AT_SYMBOL`.

[LDF 2006.12.14.] Added code for `PERCENT`.

⟨ Declare **Scan_Parse** functions 264 ⟩ ≡
  int *initialize_maps*(**void**);

See also sections 287, 291, 292, 294, 295, 296, 297, 299, and 300.

This code is used in sections 251 and 255.

265.

⟨ Define **Scan_Parse** functions 265 ⟩ ≡
  int Scan_Parse :: *initialize_maps*(**void**){ **Keyword_Type** ∗*curr_keyword*;

See also sections 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 279, 280, 281, 282, 283, 284, 285, and 288.

This code is used in section 357.

266. Punctuation. [LDF 2006.10.19.]

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword->name = "AT_SYMBOL";
curr_keyword->value = AT_SYMBOL;
curr_keyword->value_name = "AT_SYMBOL";
keyword_map["@"] = curr_keyword;
token_map[AT_SYMBOL] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "COMMA";
curr_keyword->value = COMMA;
curr_keyword->value_name = "COMMA";
keyword_map[","] = curr_keyword;
token_map[COMMA] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "PERCENT";
curr_keyword->value = PERCENT;
curr_keyword->value_name = "PERCENT";
keyword_map[","] = curr_keyword;
token_map[PERCENT] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "COLON";
curr_keyword->value = COLON;
curr_keyword->value_name = "COLON";
keyword_map[":"] = curr_keyword;
token_map[COLON] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "PERIOD";
curr_keyword->value = PERIOD;
curr_keyword->value_name = "PERIOD";
keyword_map["."] = curr_keyword;
token_map[PERIOD] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "HYPHEN";
curr_keyword->value = HYPHEN;
curr_keyword->value_name = "HYPHEN";
keyword_map["-"] = curr_keyword;
token_map[HYPHEN] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "SEMI_COLON";
curr_keyword->value = SEMI_COLON;
curr_keyword->value_name = "SEMI_COLON";
keyword_map[";"] = curr_keyword;
token_map[SEMI_COLON] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "OPEN_PARENTHESIS";
curr_keyword->value = OPEN_PARENTHESIS;
curr_keyword->value_name = "OPEN_PARENTHESIS";
keyword_map["("] = curr_keyword;
token_map[OPEN_PARENTHESIS] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "CLOSE_PARENTHESIS";
```

$curr\_keyword \rightarrow value$ = CLOSE_PARENTHESIS;
$curr\_keyword \rightarrow value\_name$ = "CLOSE_PARENTHESIS";
$keyword\_map$[")"] = $curr\_keyword$;
$token\_map$[CLOSE_PARENTHESIS] = $curr\_keyword \rightarrow value\_name$;
$curr\_keyword$ = **new Keyword_Type**;
$curr\_keyword \rightarrow name$ = "OPEN_BRACKET";
$curr\_keyword \rightarrow value$ = OPEN_BRACKET;
$curr\_keyword \rightarrow value\_name$ = "OPEN_BRACKET";
$keyword\_map$["["] = $curr\_keyword$;
$token\_map$[OPEN_BRACKET] = $curr\_keyword \rightarrow value\_name$;
$curr\_keyword$ = **new Keyword_Type**;
$curr\_keyword \rightarrow name$ = "CLOSE_BRACKET";
$curr\_keyword \rightarrow value$ = CLOSE_BRACKET;
$curr\_keyword \rightarrow value\_name$ = "CLOSE_BRACKET";
$keyword\_map$["]"] = $curr\_keyword$;
$token\_map$[CLOSE_BRACKET] = $curr\_keyword \rightarrow value\_name$;

267. Assignment. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section with code for the following keywords: `ASSIGN`, `PLUS_ASSIGN`, `MINUS_ASSIGN`, `TIMES_ASSIGN`, and `DIVIDE_ASSIGN`.

[LDF 2006.11.14.] Added code for `AND_ASSIGN`, `OR_ASSIGN`, `XOR_ASSIGN`, and `NOT_ASSIGN`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "ASSIGN";
curr_keyword→value = ASSIGN;
curr_keyword→value_name = "ASSIGN";
keyword_map["="] = curr_keyword;
token_map[ASSIGN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "PLUS_ASSIGN";
curr_keyword→value = PLUS_ASSIGN;
curr_keyword→value_name = "PLUS_ASSIGN";
keyword_map["+="] = curr_keyword;
token_map[PLUS_ASSIGN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "MINUS_ASSIGN";
curr_keyword→value = MINUS_ASSIGN;
curr_keyword→value_name = "MINUS_ASSIGN";
keyword_map["-="] = curr_keyword;
token_map[MINUS_ASSIGN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "TIMES_ASSIGN";
curr_keyword→value = TIMES_ASSIGN;
curr_keyword→value_name = "TIMES_ASSIGN";
keyword_map["*="] = curr_keyword;
token_map[TIMES_ASSIGN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "DIVIDE_ASSIGN";
curr_keyword→value = DIVIDE_ASSIGN;
curr_keyword→value_name = "DIVIDE_ASSIGN";
keyword_map["/="] = curr_keyword;
token_map[DIVIDE_ASSIGN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "AND_ASSIGN";
curr_keyword→value = AND_ASSIGN;
curr_keyword→value_name = "AND_ASSIGN";
keyword_map["&="] = curr_keyword;
token_map[AND_ASSIGN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "OR_ASSIGN";
curr_keyword→value = OR_ASSIGN;
curr_keyword→value_name = "OR_ASSIGN";
keyword_map["|="] = curr_keyword;
token_map[OR_ASSIGN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
```

```
curr_keyword→name = "XOR_ASSIGN";
curr_keyword→value = XOR_ASSIGN;
curr_keyword→value_name = "XOR_ASSIGN";
keyword_map["^="] = curr_keyword;
token_map[XOR_ASSIGN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "NOT_ASSIGN";
curr_keyword→value = NOT_ASSIGN;
curr_keyword→value_name = "NOT_ASSIGN";
keyword_map["!="] = curr_keyword;
token_map[NOT_ASSIGN] = curr_keyword→value_name;
```

268. Arithmetical Operations. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this section with code for PLUS, MINUS, TIMES, and DIVIDE.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "PLUS";
curr_keyword→value = PLUS;
curr_keyword→value_name = "PLUS";
keyword_map["+"] = curr_keyword;
token_map[PLUS] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "MINUS";
curr_keyword→value = MINUS;
curr_keyword→value_name = "MINUS";
keyword_map["-"] = curr_keyword;
token_map[MINUS] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "TIMES";
curr_keyword→value = TIMES;
curr_keyword→value_name = "TIMES";
keyword_map["*"] = curr_keyword;
token_map[TIMES] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "DIVIDE";
curr_keyword→value = DIVIDE;
curr_keyword→value_name = "DIVIDE";
keyword_map["/"] = curr_keyword;
token_map[DIVIDE] = curr_keyword→value_name;
```

269. Types with values. [LDF 2006.10.19.]

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "STRING";
curr_keyword→value = STRING;
curr_keyword→value_name = "STRING";
keyword_map["STRING"] = curr_keyword;
token_map[STRING] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "INTEGER";
curr_keyword→value = INTEGER;
curr_keyword→value_name = "INTEGER";
keyword_map["INTEGER"] = curr_keyword;
token_map[INTEGER] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "FLOAT";
curr_keyword→value = FLOAT;
curr_keyword→value_name = "FLOAT";
keyword_map["FLOAT"] = curr_keyword;
token_map[FLOAT] = curr_keyword→value_name;
```

270. Control. [LDF 2006.10.19.]

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "start";
curr_keyword→value = START;
curr_keyword→value_name = "START";
keyword_map["start"] = curr_keyword;
token_map[START] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "end";
curr_keyword→value = END;
curr_keyword→value_name = "END";
keyword_map["end"] = curr_keyword;
token_map[END] = curr_keyword→value_name;
```

271. Queries. [LDF 2006.10.19.]

Log

[2006.11.16.] Added code for `STRING_DECLARATOR`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "query";
curr_keyword→value = QUERY_DECLARATOR;
curr_keyword→value_name = "QUERY_DECLARATOR";
keyword_map["query"] = curr_keyword;
token_map[QUERY_DECLARATOR] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "local";
curr_keyword→value = LOCAL;
curr_keyword→value_name = "LOCAL";
keyword_map["local"] = curr_keyword;
token_map[LOCAL] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "remote";
curr_keyword→value = REMOTE;
curr_keyword→value_name = "REMOTE";
keyword_map["remote"] = curr_keyword;
token_map[REMOTE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "database";
curr_keyword→value = DATABASE;
curr_keyword→value_name = "DATABASE";
keyword_map["database"] = curr_keyword;
token_map[DATABASE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "server";
curr_keyword→value = SERVER;
curr_keyword→value_name = "SERVER";
keyword_map["server"] = curr_keyword;
token_map[SERVER] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "string";
curr_keyword→value = STRING_DECLARATOR;
curr_keyword→value_name = "STRING_DECLARATOR";
keyword_map["string"] = curr_keyword;
token_map[STRING_DECLARATOR] = curr_keyword→value_name;
```

272. Match types. [LDF 2006.12.11.]

Log

[2006.12.11.] Added this section with code for `LIKE` and `FREETEXT`.

[LDF 2006.12.12.] Added code for `CONTAINS`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "contains";
curr_keyword→value = CONTAINS;
curr_keyword→value_name = "CONTAINS";
keyword_map["contains"] = curr_keyword;
token_map[CONTAINS] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "freetext";
curr_keyword→value = FREETEXT;
curr_keyword→value_name = "FREETEXT";
keyword_map["freetext"] = curr_keyword;
token_map[FREETEXT] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "like";
curr_keyword→value = LIKE;
curr_keyword→value_name = "LIKE";
keyword_map["like"] = curr_keyword;
token_map[LIKE] = curr_keyword→value_name;
```

273. Datasources. [LDF 2006.12.08.]

Log

[2006.11.16.] Added this section with code for `DATASOURCE_DECLARATOR`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "datasource";
curr_keyword→value = DATASOURCE_DECLARATOR;
curr_keyword→value_name = "DATASOURCE_DECLARATOR";
keyword_map["datasource"] = curr_keyword;
token_map[DATASOURCE_DECLARATOR] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "dbt";
curr_keyword→value = DBT;
curr_keyword→value_name = "DBT";
keyword_map["dbt"] = curr_keyword;
token_map[DBT] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "gbv_gvk";
curr_keyword→value = GBV_GVK;
curr_keyword→value_name = "GBV_GVK";
keyword_map["gbv_gvk"] = curr_keyword;
token_map[GBV_GVK] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "timms";
curr_keyword→value = TIMMS;
curr_keyword→value_name = "TIMMS";
keyword_map["timms"] = curr_keyword;
token_map[TIMMS] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "file";
curr_keyword→value = DATASOURCE_FILE;
curr_keyword→value_name = "DATASOURCE_FILE";
keyword_map["file"] = curr_keyword;
token_map[DATASOURCE_FILE] = curr_keyword→value_name;
```

274. Datetime. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this section with code for `DATETIME_DECLARATOR`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "datetime";
curr_keyword→value = DATETIME_DECLARATOR;
curr_keyword→value_name = "DATETIME_DECLARATOR";
keyword_map["datetime"] = curr_keyword;
token_map[DATETIME_DECLARATOR] = curr_keyword→value_name;
```

275. Datetimes. [LDF 2006.12.15.]

Log

[2006.12.15.] Added code for `DAY`, `MONTH`, `YEAR`, `HOUR`, `MINUTE`, and `SECOND`.

[LDF 2006.12.18.] Added code for `YEAR_RANGE_BEGIN` and `YEAR_RANGE_END`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "year_range_begin";
curr_keyword→value = YEAR_RANGE_BEGIN;
curr_keyword→value_name = "YEAR_RANGE_BEGIN";
keyword_map["year_range_begin"] = curr_keyword;
token_map[YEAR_RANGE_BEGIN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "year_range_end";
curr_keyword→value = YEAR_RANGE_END;
curr_keyword→value_name = "YEAR_RANGE_END";
keyword_map["year_range_end"] = curr_keyword;
token_map[YEAR_RANGE_END] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "year";
curr_keyword→value = YEAR;
curr_keyword→value_name = "YEAR";
keyword_map["year"] = curr_keyword;
token_map[YEAR] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "month";
curr_keyword→value = MONTH;
curr_keyword→value_name = "MONTH";
keyword_map["month"] = curr_keyword;
token_map[MONTH] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "day";
curr_keyword→value = DAY;
curr_keyword→value_name = "DAY";
keyword_map["day"] = curr_keyword;
token_map[DAY] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "hour";
curr_keyword→value = HOUR;
curr_keyword→value_name = "HOUR";
keyword_map["hour"] = curr_keyword;
token_map[HOUR] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "minute";
curr_keyword→value = MINUTE;
curr_keyword→value_name = "MINUTE";
keyword_map["minute"] = curr_keyword;
token_map[MINUTE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "second";
```

```
curr_keyword→value = SECOND;
curr_keyword→value_name = "SECOND";
keyword_map["second"] = curr_keyword;
token_map[SECOND] = curr_keyword→value_name;
```

276. Predicates and Control Structures. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this section with code for `IF`, `ELSE`, `ELIF`, `FI`, `FOR`, `DO`, `WHILE`, `AND`, `OR`, `XOR`, `NOT`, `AND_NOT`, `OR_NOT`, and `XOR_NOT`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "if";
curr_keyword→value = IF;
curr_keyword→value_name = "IF";
keyword_map["if"] = curr_keyword;
token_map[IF] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "else";
curr_keyword→value = ELSE;
curr_keyword→value_name = "ELSE";
keyword_map["else"] = curr_keyword;
token_map[ELSE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "elif";
curr_keyword→value = ELIF;
curr_keyword→value_name = "ELIF";
keyword_map["elif"] = curr_keyword;
token_map[ELIF] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "fi";
curr_keyword→value = FI;
curr_keyword→value_name = "FI";
keyword_map["fi"] = curr_keyword;
token_map[FI] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "for";
curr_keyword→value = FOR;
curr_keyword→value_name = "FOR";
keyword_map["for"] = curr_keyword;
token_map[FOR] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "do";
curr_keyword→value = DO;
curr_keyword→value_name = "DO";
keyword_map["do"] = curr_keyword;
token_map[DO] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "while";
curr_keyword→value = WHILE;
curr_keyword→value_name = "WHILE";
keyword_map["while"] = curr_keyword;
token_map[WHILE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "and";
```

```
curr_keyword->value = AND;
curr_keyword->value_name = "AND";
keyword_map["and"] = curr_keyword;
keyword_map["&"] = curr_keyword;
token_map[AND] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "or";
curr_keyword->value = OR;
curr_keyword->value_name = "OR";
keyword_map["or"] = curr_keyword;
keyword_map["|"] = curr_keyword;
token_map[OR] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "xor";
curr_keyword->value = XOR;
curr_keyword->value_name = "XOR";
keyword_map["xor"] = curr_keyword;
keyword_map["^"] = curr_keyword;
token_map[XOR] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "not";
curr_keyword->value = NOT;
curr_keyword->value_name = "NOT";
keyword_map["!"] = curr_keyword;
token_map[NOT] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "AND_NOT";
curr_keyword->value = AND_NOT;
curr_keyword->value_name = "AND_NOT";
keyword_map["&!"] = curr_keyword;
token_map[AND_NOT] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "OR_NOT";
curr_keyword->value = OR_NOT;
curr_keyword->value_name = "OR_NOT";
keyword_map["|!"] = curr_keyword;
token_map[OR_NOT] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "XOR_NOT";
curr_keyword->value = XOR_NOT;
curr_keyword->value_name = "XOR_NOT";
keyword_map["^!"] = curr_keyword;
token_map[XOR_NOT] = curr_keyword->value_name;
```

277. Fields. [LDF 2006.10.19.]

Log

[LDF 2006.12.05.] Added code for CREATOR.

[LDF 2006.12.12.] Added code for MAIN_CANONICAL_TITLE.

[LDF 2006.12.12.] Added code to account for the tokens that refer to additional tables in the OAI database (dc_test).

[LDF 2006.12.12.] Added code to account for the tokens that refer to additional tables in the PICA database (PICA_DB).

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "access_number";
curr_keyword→value = ACCESS_NUMBER;
curr_keyword→value_name = "ACCESS_NUMBER";
keyword_map["access_number"] = curr_keyword;
token_map[ACCESS_NUMBER] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "author";
curr_keyword→value = AUTHOR;
curr_keyword→value_name = "AUTHOR";
keyword_map["author"] = curr_keyword;
token_map[AUTHOR] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "bibliographic_type";
curr_keyword→value = BIBLIOGRAPHIC_TYPE;
curr_keyword→value_name = "BIBLIOGRAPHIC_TYPE";
keyword_map["bibliographic_type"] = curr_keyword;
token_map[BIBLIOGRAPHIC_TYPE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "call_number";
curr_keyword→value = CALL_NUMBER;
curr_keyword→value_name = "CALL_NUMBER";
keyword_map["call_number"] = curr_keyword;
token_map[CALL_NUMBER] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "classification";
curr_keyword→value = CLASSIFICATION;
curr_keyword→value_name = "CLASSIFICATION";
keyword_map["classification"] = curr_keyword;
token_map[CLASSIFICATION] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "company";
curr_keyword→value = COMPANY;
curr_keyword→value_name = "COMPANY";
keyword_map["company"] = curr_keyword;
token_map[COMPANY] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "content_summary";
curr_keyword→value = CONTENT_SUMMARY;
```

```
curr_keyword->value_name = "CONTENT_SUMMARY";
keyword_map["content_summary"] = curr_keyword;
token_map[CONTENT_SUMMARY] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "contributor";
curr_keyword->value = CONTRIBUTOR;
curr_keyword->value_name = "CONTRIBUTOR";
keyword_map["contributor"] = curr_keyword;
token_map[CONTRIBUTOR] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "creator";
curr_keyword->value = CREATOR;
curr_keyword->value_name = "CREATOR";
keyword_map["creator"] = curr_keyword;
token_map[CREATOR] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "database_provider";
curr_keyword->value = DATABASE_PROVIDER;
curr_keyword->value_name = "DATABASE_PROVIDER";
keyword_map["database_provider"] = curr_keyword;
token_map[DATABASE_PROVIDER] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "description";
curr_keyword->value = DESCRIPTION;
curr_keyword->value_name = "DESCRIPTION";
keyword_map["description"] = curr_keyword;
token_map[DESCRIPTION] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "exemplar_production_number";
curr_keyword->value = EXEMPLAR_PRODUCTION_NUMBER;
curr_keyword->value_name = "EXEMPLAR_PRODUCTION_NUMBER";
keyword_map["exemplar_production_number"] = curr_keyword;
token_map[EXEMPLAR_PRODUCTION_NUMBER] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "identifier";
curr_keyword->value = IDENTIFIER;
curr_keyword->value_name = "IDENTIFIER";
keyword_map["identifier"] = curr_keyword;
token_map[IDENTIFIER] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "institution";
curr_keyword->value = INSTITUTION;
curr_keyword->value_name = "INSTITUTION";
keyword_map["institution"] = curr_keyword;
token_map[INSTITUTION] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
curr_keyword->name = "language";
curr_keyword->value = LANGUAGE;
curr_keyword->value_name = "LANGUAGE";
keyword_map["language"] = curr_keyword;
token_map[LANGUAGE] = curr_keyword->value_name;
curr_keyword = new Keyword_Type;
```

```
curr_keyword→name = "main_canonical_title";
curr_keyword→value = MAIN_CANONICAL_TITLE;
curr_keyword→value_name = "MAIN_CANONICAL_TITLE";
keyword_map["main_canonical_title"] = curr_keyword;
token_map[MAIN_CANONICAL_TITLE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "permutation_pattern";
curr_keyword→value = PERMUTATION_PATTERN;
curr_keyword→value_name = "PERMUTATION_PATTERN";
keyword_map["permutation_pattern"] = curr_keyword;
token_map[PERMUTATION_PATTERN] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "person";
curr_keyword→value = PERSON;
curr_keyword→value_name = "PERSON";
keyword_map["person"] = curr_keyword;
token_map[PERSON] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "physical_description";
curr_keyword→value = PHYSICAL_DESCRIPTION;
curr_keyword→value_name = "PHYSICAL_DESCRIPTION";
keyword_map["physical_description"] = curr_keyword;
token_map[PHYSICAL_DESCRIPTION] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "publisher";
curr_keyword→value = PUBLISHER;
curr_keyword→value_name = "PUBLISHER";
keyword_map["publisher"] = curr_keyword;
token_map[PUBLISHER] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "record";
curr_keyword→value = RECORD;
curr_keyword→value_name = "RECORD";
keyword_map["record"] = curr_keyword;
token_map[RECORD] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "remote_access";
curr_keyword→value = REMOTE_ACCESS;
curr_keyword→value_name = "REMOTE_ACCESS";
keyword_map["remote_access"] = curr_keyword;
token_map[REMOTE_ACCESS] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "rights";
curr_keyword→value = RIGHTS;
curr_keyword→value_name = "RIGHTS";
keyword_map["rights"] = curr_keyword;
token_map[RIGHTS] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "source";
curr_keyword→value = SOURCE;
curr_keyword→value_name = "SOURCE";
keyword_map["source"] = curr_keyword;
```

token_map[SOURCE] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "subject";
curr_keyword→*value* = SUBJECT;
curr_keyword→*value_name* = "SUBJECT";
keyword_map["subject"] = *curr_keyword*;
token_map[SUBJECT] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "superordinate_entities";
curr_keyword→*value* = SUPERORDINATE_ENTITIES;
curr_keyword→*value_name* = "SUPERORDINATE_ENTITIES";
keyword_map["superordinate_entities"] = *curr_keyword*;
token_map[SUPERORDINATE_ENTITIES] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "title";
curr_keyword→*value* = TITLE;
curr_keyword→*value_name* = "TITLE";
keyword_map["title"] = *curr_keyword*;
token_map[TITLE] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "type";
curr_keyword→*value* = TYPE;
curr_keyword→*value_name* = "TYPE";
keyword_map["type"] = *curr_keyword*;
token_map[TYPE] = *curr_keyword*→*value_name*;

278. Keywords for database table columns. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

---

279. Generic. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "id";
curr_keyword→*value* = ID;
curr_keyword→*value_name* = "ID";
keyword_map["id"] = *curr_keyword*;
token_map[ID] = *curr_keyword*→*value_name*;

280. Names. [2006.12.07.]

Log

[2006.12.07.] Added this section with code for `GIVEN_NAME`, `PREFIX`, and `SURNAME`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "given_name";
curr_keyword→value = GIVEN_NAME;
curr_keyword→value_name = "GIVEN_NAME";
keyword_map["given_name"] = curr_keyword;
token_map[GIVEN_NAME] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "prefix";
curr_keyword→value = PREFIX;
curr_keyword→value_name = "PREFIX";
keyword_map["prefix"] = curr_keyword;
token_map[PREFIX] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "surname";
curr_keyword→value = SURNAME;
curr_keyword→value_name = "SURNAME";
keyword_map["surname"] = curr_keyword;
token_map[SURNAME] = curr_keyword→value_name;
```

281. Records. [2006.12.07.]

Log

[2006.12.14.] Added this section.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "eln_original_entry";
curr_keyword→*value* = ELN_ORIGINAL_ENTRY;
curr_keyword→*value_name* = "ELN_ORIGINAL_ENTRY";
keyword_map["eln_original_entry"] = *curr_keyword*;
token_map[ELN_ORIGINAL_ENTRY] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "eln_most_recent_change";
curr_keyword→*value* = ELN_MOST_RECENT_CHANGE;
curr_keyword→*value_name* = "ELN_MOST_RECENT_CHANGE";
keyword_map["eln_most_recent_change"] = *curr_keyword*;
token_map[ELN_MOST_RECENT_CHANGE] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "eln_status_change";
curr_keyword→*value* = ELN_STATUS_CHANGE;
curr_keyword→*value_name* = "ELN_STATUS_CHANGE";
keyword_map["eln_status_change"] = *curr_keyword*;
token_map[ELN_STATUS_CHANGE] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "identification_number";
curr_keyword→*value* = IDENTIFICATION_NUMBER;
curr_keyword→*value_name* = "IDENTIFICATION_NUMBER";
keyword_map["identification_number"] = *curr_keyword*;
token_map[IDENTIFICATION_NUMBER] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "date_original_entry";
curr_keyword→*value* = DATE_ORIGINAL_ENTRY;
curr_keyword→*value_name* = "DATE_ORIGINAL_ENTRY";
keyword_map["date_original_entry"] = *curr_keyword*;
token_map[DATE_ORIGINAL_ENTRY] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "date_most_recent_change";
curr_keyword→*value* = DATE_MOST_RECENT_CHANGE;
curr_keyword→*value_name* = "DATE_MOST_RECENT_CHANGE";
keyword_map["date_most_recent_change"] = *curr_keyword*;
token_map[DATE_MOST_RECENT_CHANGE] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "date_status_change";
curr_keyword→*value* = DATE_STATUS_CHANGE;
curr_keyword→*value_name* = "DATE_STATUS_CHANGE";
keyword_map["date_status_change"] = *curr_keyword*;
token_map[DATE_STATUS_CHANGE] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "source_id";
curr_keyword→*value* = SOURCE_ID;

curr_keyword→*value_name* = "SOURCE_ID";
keyword_map["source_id"] = *curr_keyword*;
token_map[SOURCE_ID] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "year_appearance_begin";
curr_keyword→*value* = YEAR_APPEARANCE_BEGIN;
curr_keyword→*value_name* = "YEAR_APPEARANCE_BEGIN";
keyword_map["year_appearance_begin"] = *curr_keyword*;
token_map[YEAR_APPEARANCE_BEGIN] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "year_appearance_end";
curr_keyword→*value* = YEAR_APPEARANCE_END;
curr_keyword→*value_name* = "YEAR_APPEARANCE_END";
keyword_map["year_appearance_end"] = *curr_keyword*;
token_map[YEAR_APPEARANCE_END] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "year_appearance_rak_wb";
curr_keyword→*value* = YEAR_APPEARANCE_RAK_WB;
curr_keyword→*value_name* = "YEAR_APPEARANCE_RAK_WB";
keyword_map["year_appearance_rak_wb"] = *curr_keyword*;
token_map[YEAR_APPEARANCE_RAK_WB] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "year_appearance_original";
curr_keyword→*value* = YEAR_APPEARANCE_ORIGINAL;
curr_keyword→*value_name* = "YEAR_APPEARANCE_ORIGINAL";
keyword_map["year_appearance_original"] = *curr_keyword*;
token_map[YEAR_APPEARANCE_ORIGINAL] = *curr_keyword*→*value_name*;

282. Variables. [LDF Undated.]

⟨ Define **Scan_Parse** functions 265 ⟩ +≡
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "VARIABLE";
curr_keyword→*value* = VARIABLE;
curr_keyword→*value_name* = "VARIABLE";
keyword_map["VARIABLE"] = *curr_keyword*;
token_map[VARIABLE] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "VARIABLE_TEXT_SEGMENT";
curr_keyword→*value* = VARIABLE_TEXT_SEGMENT;
curr_keyword→*value_name* = "VARIABLE_TEXT_SEGMENT";
keyword_map["VARIABLE_TEXT_SEGMENT"] = *curr_keyword*;
token_map[VARIABLE_TEXT_SEGMENT] = *curr_keyword*→*value_name*;

283.

Log

[LDF 2006.11.16.] Added code for `STRING_TYPE`.

[LDF 2006.12.08.] Added code for `DATASOURCE_TYPE`.

[LDF 2006.12.15.] Added code for `DATETIME_TYPE`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "NULL_TYPE";
curr_keyword→value = NULL_TYPE;
curr_keyword→value_name = "NULL_TYPE";
keyword_map["NULL_TYPE"] = curr_keyword;
token_map[NULL_TYPE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "DATASOURCE_TYPE";
curr_keyword→value = DATASOURCE_TYPE;
curr_keyword→value_name = "DATASOURCE_TYPE";
keyword_map["DATASOURCE_TYPE"] = curr_keyword;
token_map[DATASOURCE_TYPE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "DATETIME_TYPE";
curr_keyword→value = DATETIME_TYPE;
curr_keyword→value_name = "DATETIME_TYPE";
keyword_map["DATETIME_TYPE"] = curr_keyword;
token_map[DATETIME_TYPE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "QUERY_TYPE";
curr_keyword→value = QUERY_TYPE;
curr_keyword→value_name = "QUERY_TYPE";
keyword_map["QUERY_TYPE"] = curr_keyword;
token_map[QUERY_TYPE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "STRING_TYPE";
curr_keyword→value = STRING_TYPE;
curr_keyword→value_name = "STRING_TYPE";
keyword_map["STRING_TYPE"] = curr_keyword;
token_map[STRING_TYPE] = curr_keyword→value_name;
```

284. Commands. [LDF 2006.11.13.]
[LDF 2006.10.19.]

Log

[LDF 2006.11.13.] Added code for `CLEAR` and `SHOW`.

[LDF 2006.11.13.] Added code for `CLEAR` and `SHOW`.

[LDF 2006.11.13.] Added code for `MESSAGE`, `ERRMESSAGE`, and `PAUSE`.

[LDF 2006.11.16.] Added code for `TEX`, `SQL`, `OAI`, and `PICA`.

[LDF 2006.11.23.] Added code for `OUTPUT`.

⟨ Define **Scan_Parse** functions 265 ⟩ +≡

```
curr_keyword = new Keyword_Type;
curr_keyword→name = "clear";
curr_keyword→value = CLEAR;
curr_keyword→value_name = "CLEAR";
keyword_map["clear"] = curr_keyword;
token_map[CLEAR] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "errmessage";
curr_keyword→value = ERRMESSAGE;
curr_keyword→value_name = "ERRMESSAGE";
keyword_map["errmessage"] = curr_keyword;
token_map[ERRMESSAGE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "message";
curr_keyword→value = MESSAGE;
curr_keyword→value_name = "MESSAGE";
keyword_map["message"] = curr_keyword;
token_map[MESSAGE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "pause";
curr_keyword→value = PAUSE;
curr_keyword→value_name = "PAUSE";
keyword_map["pause"] = curr_keyword;
token_map[PAUSE] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "show";
curr_keyword→value = SHOW;
curr_keyword→value_name = "SHOW";
keyword_map["show"] = curr_keyword;
token_map[SHOW] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "tex";
curr_keyword→value = TEX;
curr_keyword→value_name = "TEX";
keyword_map["tex"] = curr_keyword;
token_map[TEX] = curr_keyword→value_name;
curr_keyword = new Keyword_Type;
curr_keyword→name = "sql";
curr_keyword→value = SQL;
```

curr_keyword→*value_name* = "SQL";
keyword_map["sql"] = *curr_keyword*;
token_map[SQL] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "oai";
curr_keyword→*value* = OAI;
curr_keyword→*value_name* = "OAI";
keyword_map["oai"] = *curr_keyword*;
token_map[OAI] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "pica";
curr_keyword→*value* = PICA;
curr_keyword→*value_name* = "PICA";
keyword_map["pica"] = *curr_keyword*;
token_map[PICA] = *curr_keyword*→*value_name*;
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "output";
curr_keyword→*value* = OUTPUT;
curr_keyword→*value_name* = "OUTPUT";
keyword_map["output"] = *curr_keyword*;
token_map[OUTPUT] = *curr_keyword*→*value_name*;

285. Control. [LDF 2006.10.19.]

⟨ Define **Scan_Parse** functions 265 ⟩ +≡
curr_keyword = **new Keyword_Type**;
curr_keyword→*name* = "terminate";
curr_keyword→*value* = TERMINATE;
curr_keyword→*value_name* = "TERMINATE";
keyword_map["terminate"] = *curr_keyword*;
token_map[TERMINATE] = *curr_keyword*→*value_name*;
return 0; } /* End of **Scan_Parse** :: *initialize_maps* definition. */

286. Show *keyword_map*. [LDF 2006.10.19.]

287.

⟨ Declare **Scan_Parse** functions 264 ⟩ +≡
int *show_keyword_map*(**Scanner_Type** **scanner_node*);

288.

```
⟨ Define Scan_Parse functions 265 ⟩ +≡
  int Scan_Parse::show_keyword_map(Scanner_Type *scanner_node)
  {
    stringstream temp_strm;
    temp_strm << "Showing␣‘keyword_map’:" << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    if (scanner_node) scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
    for (map<string, Keyword_Type *>::const_iterator iter = keyword_map.begin();
          iter != keyword_map.end(); ++iter) {
      temp_strm << "name:␣␣" << iter->second->name << ",␣value:␣␣" << token_map[iter->second->value] <<
          "␣(" << iter->second->value << ")␣" << ",␣constant␣name:␣␣" << iter->second->value_name <<
          endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      if (scanner_node) scanner_node->log_strm << temp_strm.str();
      temp_strm.str("");
    }    /* for */
    return 0;
  }    /* End of Scan_Parse::show_keyword_map definition. */
```

289. Parser rule functions. [LDF 2006.10.31.]

These functions are used in the parser rules. They are defined in other files. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

---

290. Functions for variables. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this section.

---

291. *variable_func.* [LDF 2006.11.02.]

Log

[2006.11.02.] Added this declaration.

⟨ Declare **Scan_Parse** functions 264 ⟩ +≡
void ∗*variable_func*(**void** ∗*v*, **char** ∗*name*);

292. Functions for declarations. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this declaration.

[LDF 2006.11.01.] Added the **char** ∗*name* argument.

[LDF 2006.11.02.] Changed the return value from **int** to **void** ∗.

⟨ Declare **Scan_Parse** functions 264 ⟩ +≡
Id_Node *declare_variable_func*(**void** ∗*v*, **const unsigned short** *type*, **char** ∗*name*);

293. Functions for assignments. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this section.

---

294. *query_assignment_func_0.* [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this declaration.

⟨ Declare **Scan_Parse** functions 264 ⟩ +≡
void ∗*query_assignment_func_0*(**void** ∗*v*, **void** ∗*object*, **unsigned int** *assignment_type*, **unsigned int** *arg_0*, **unsigned int** *arg_1*, **void** ∗*value*, **bool** *negate*);

295. *query_assignment_func_1.* [LDF 2006.12.19.]

Log

[LDF 2006.12.19.] Added this declaration.

⟨ Declare **Scan_Parse** functions 264 ⟩ +≡
int *query_assignment_func_1*(**Scanner_Node** *scanner_node*, **Id_Node** &*curr_id_node*, **void** ∗&*field_specifier*, **int** *assignment_operator*, **int** *negation_optional*, **int** *match_term_optional*, **void** ∗&*v*, **int** *type*);

296. *datetime_assignment_func_0.* [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this declaration.

⟨ Declare **Scan_Parse** functions 264 ⟩ +≡
int *datetime_assignment_func_0* (**void** ∗*v*, **Date_Time_Node** &*curr_date_time_node*, **int** *specifier*, **int** *op*, **void** ∗*val*, **int** *type*);

297. *datetime_assignment_func_1.* [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this declaration.

⟨ Declare **Scan_Parse** functions 264 ⟩ +≡
int *datetime_assignment_func_1* (**void** ∗*v*, **Date_Time_Node** &*d*, **int** *op*, **void** ∗&*vec*);

298. Functions for Group Statements. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

---

299. Start local database query function. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this function declaration.

[LDF 2006.11.01.] Changed the name of this function from **Scan_Parse** :: *start_local_query_func* to **Scan_Parse** :: *start_local_database_query_func*.

⟨ Declare **Scan_Parse** functions 264 ⟩ +≡
int *start_local_database_query_func*(**void** ∗);

300. End query function. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this function declaration.

[LDF 2006.11.01.] Changed the name of this function from **Scan_Parse** :: *end_local_query_func* to **Scan_Parse** :: *end_query_func*.

⟨ Declare **Scan_Parse** functions 264 ⟩ +≡
  int *end_query_func*(**void** ∗);

301. The scanning function *yylex*. [LDF 2006.10.17.]

Log

[LDF 2006.11.13.] Added code for handling '+'.

⟨ Declare *yylex* 301 ⟩ ≡
  int *yylex*(YYSTYPE ∗ *value*, **void** ∗*parameter*);

See also section 361.

This code is used in sections 358 and 643.

302.

⟨ Define *yylex* 302 ⟩ ≡
  int *yylex*(YYSTYPE ∗ *value*, **void** ∗*parameter*){
#if 0 /∗ 1 ∗/
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    using namespace Scan_Parse;
    stringstream *temp_strm*; *stack* < **unsigned short** > *state_stack*;
    unsigned short *state* = NULL_STATE;
    Scanner_Type ∗*scanner_node* = **static_cast**⟨**Scanner_Type** ∗⟩(*parameter*);

See also sections 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, and 350.

This code is used in section 357.

303. "Fake" a token. If *scanner_node→token_stack* isn't empty, extract the token to be returned and its semantic value from the **Token_Type** object at the top of the stack, pop it from the stack, and return immediately, without reading anything from the input stream. This will happen repeatedly, until *scanner_node→token_stack* is empty. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  if (scanner_node→token_stack.size() > 0) {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "In␣‘yylex’:␣␣‘scanner_node->token_stack.size()␣==␣" ≪
        scanner_node→token_stack.size() ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Token_Type curr_token = scanner_node→token_stack.top();
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "‘curr_token.type’␣==␣" ≪ token_map[curr_token.type] ≪ "␣(" ≪ curr_token.type ≪
        ")." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    if (curr_token.type ≡ STRING) strcpy(value→string_value, curr_token.value.string_value);
    else if (curr_token.type ≡ INTEGER) value→int_value = curr_token.value.int_value;
    else if (curr_token.type ≡ FLOAT) value→float_value = curr_token.value.float_value;
    else {
      value→pointer_value = curr_token.value.pointer_value;
      curr_token.value.pointer_value = 0;
    }
    scanner_node→token_stack.pop();
    return curr_token.type;
  }   /* if (scanner_node→token_stack.size() > 0) */
  else {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "In␣‘yylex’:␣␣‘scanner_node->token_stack.size()␣==␣0." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
  }
```

304. Open input file. [LDF 2006.10.17.]

⟨ Define *yylex* 302 ⟩ +≡

```
#ifdef DEBUG_OUTPUT
  temp_strm << "Entering␣'yylex'." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm << temp_strm.str();
  temp_strm.str("");
#endif
#if 0   /* 1 */
#ifdef DEBUG_OUTPUT
  temp_strm << "'scanner_node->in_filename'␣==␣" << scanner_node→in_filename << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm << temp_strm.str();
  temp_strm.str("");
#endif
#endif
```

305. Main loop. [LDF 2006.10.17.]

Log

[LDF 2006.11.01.] Changed **string** *curr_keyword* to *curr_id*.

[LDF 2006.11.13.] Added *next_char*.

```
⟨ Define yylex 302 ⟩ +≡
  char curr_char = 0;
  char next_char = 0;
  string curr_string = "";
  string curr_id = "";
  int curr_value = 0;
  string curr_integer_str = "";
  string curr_float_str = ""; while (curr_char ≠ EOF) { curr_char = scanner_node→in_strm.get();
#ifdef DEBUG_OUTPUT
  temp_strm ≪ curr_char;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

306. Comments. '#'. Discard characters until end of line. # can occur in strings, however. [LDF 2006.10.17.]

Log

[LDF 2006.10.31.] Changed comment character from % to #.

```
⟨ Define yylex 302 ⟩ +≡
  if (curr_char ≡ '#' ∧ (state ≡ NULL_STATE ∨ state ≡ COLLECTING_ID ∨ state ≡
      COLLECTING_INTEGER ∨ state ≡ COLLECTING_FLOAT)) {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "***␣Discarding␣comment.␣␣***" ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    while (¬(curr_char ≡ '\n' ∨ curr_char ≡ EOF)) {
      curr_char = scanner_node→in_strm.get();
      if (curr_char ≡ EOF) goto FINISH;
    }    /* Inner while */
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "***␣Finished␣discarding␣comment.␣␣***" ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
```

```
#endif
  }    /* if (curr_char ≡ '#') (Discarding comment) */
```

307. Double-quote symbol. '"'. [LDF 2006.10.17.]

```
⟨ Define yylex 302 ⟩ +≡
  else if (curr_char ≡ '"') {
```

308. *state* ≡ COLLECTING_STRING [LDF 2006.10.17.]

```
⟨ Define yylex 302 ⟩ +≡
  if (state ≡ COLLECTING_STRING) {
    state = state_stack.top();
    state_stack.pop();
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "***␣Finished␣collecting␣string." ≪ endl ≪ "‘curr_string’␣==␣" ≪
        curr_string.c_str() ≪ endl ≪ "Exiting␣‘yylex’␣successfully␣with␣return␣value␣" ≪
        keyword_map["STRING"]→value_name ≪ "␣(" ≪ keyword_map["STRING"]→value ≪ ")" ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    strcpy(value→string_value, curr_string.c_str());
    curr_string = "";
    return STRING;
  }    /* if (state ≡ COLLECTING_STRING) */
```

309. *state* ≡ NULL_STATE. Start collecting string. [LDF 2006.10.18.]

⟨ Define *yylex* 302 ⟩ +≡
```
  else
  if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "***␣Starting␣to␣collect␣string.␣␣***" ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    state_stack.push(state);
    state = COLLECTING_STRING;
    curr_string = "";
  }
  else if (state ≡ COLLECTING_ID) {
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "***␣Finished␣collecting␣keyword.␣␣" ≪ "‘curr_id’␣==␣" ≪ curr_id ≪
      endl ≪ "***" ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

310. Look up *curr_id* in *keyword_map* and possibly *id_map*. [LDF 2006.10.19.]

⟨ Define *yylex* 302 ⟩ +≡
```
  ⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩
  }   /* else if (state ≡ COLLECTING_ID) */
```

311. *state* ≡ COLLECTING_INTEGER. [LDF 2006.11.02.]

⟨ Define *yylex* 302 ⟩ +≡
```
  else
    if (state ≡ COLLECTING_INTEGER) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣integer.␣␣" ≪ "‘curr_integer_str’␣==␣" ≪
          curr_integer_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→int_value = atoi(curr_integer_str.c_str());
      scanner_node→in_strm.unget();
      return INTEGER;
    }   /* else if (state ≡ COLLECTING_INTEGER) */
```

312. *state* ≡ COLLECTING_FLOAT. [LDF 2006.11.02.]

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (state ≡ COLLECTING_FLOAT) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣float.␣␣" ≪ "‘curr_float_str’␣==␣" ≪
          curr_float_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→float_value = atof(curr_float_str.c_str());
      scanner_node→in_strm.unget();
      return FLOAT;
    }   /* else if (state ≡ COLLECTING_FLOAT) */
```

313. Any other state (shouldn't occur.) [LDF 2006.11.02.]

```
⟨ Define yylex 302 ⟩ +≡
  else {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "***␣Starting␣to␣collect␣string.␣␣***" ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    state_stack.push(state);
    state = COLLECTING_STRING;
    curr_string = "";
  }
  }   /* else if (curr_char ≡ '"') */
```

314. Alphabetical characters and underline character. [LDF 2006.10.18.]

Log

[LDF 2006.12.08.] Now treating underline character like an alphabetical character.

[LDF 2006.12.11.] Added code for common accented characters.

```
⟨ Define yylex 302 ⟩ +≡
  else if (isalpha(curr_char) ∨ curr_char ≡ '_' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡
      '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡
      '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡
      '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡
      '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡
      '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡
      '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡
      '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '' ∨ curr_char ≡ '') {
```

315. *state* ≡ `NULL_STATE`. [LDF 2006.10.18.]

```
⟨ Define yylex 302 ⟩ +≡
  if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "***␣Read␣an␣alphabetical␣character␣" ≪
        "or␣the␣underline␣character␣in␣‘NULL_STATE’:" ≪ endl ≪
        "Entering␣state␣‘COLLECTING_ID’.␣***" ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    state = COLLECTING_ID;
    curr_id = "";
    curr_id += curr_char;
  }   /* if (state ≡ NULL_STATE) */
```

316. *state* ≡ COLLECTING_ID. [LDF 2006.10.18.]

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (state ≡ COLLECTING_ID) {
      curr_id += curr_char;
    }   /* if (state ≡ COLLECTING_ID) */
    else if (state ≡ COLLECTING_INTEGER) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣integer.␣␣" ≪ "‘curr_integer_str’␣==␣" ≪
          curr_integer_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→int_value = atoi(curr_integer_str.c_str());
      scanner_node→in_strm.unget();
      return INTEGER;
    }   /* else if (state ≡ COLLECTING_INTEGER) */
    else if (state ≡ COLLECTING_FLOAT) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣float.␣␣" ≪ "‘curr_float_str’␣==␣" ≪
          curr_float_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→float_value = atof(curr_float_str.c_str());
      scanner_node→in_strm.unget();
      return FLOAT;
    }   /* else if (state ≡ COLLECTING_FLOAT) */
```

317. *state* ≡ COLLECTING_STRING. [LDF 2006.10.18.]

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (state ≡ COLLECTING_STRING) {
      curr_string += curr_char;
    }    /* if (state ≡ COLLECTING_STRING) */
  }    /* else if (isalpha(curr_char) ∨ curr_char ≡ '_') */
```

318. Digits. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else if (isdigit(curr_char)) {
  if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "***␣Read␣a␣digit␣in␣‘NULL_STATE’.␣␣" ≪
        "Entering␣state␣‘COLLECTING_INTEGER’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    curr_integer_str += curr_char;
    state = COLLECTING_INTEGER;
    continue;
  }    /* if (state ≡ NULL_STATE) */
```

319. *state* ≡ COLLECTING_ID. [LDF 2006.10.18.]

```
⟨ Define yylex 302 ⟩ +≡
  else if (state ≡ COLLECTING_ID) {
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "***␣Finished␣collecting␣keyword.␣␣" ≪ "‘curr_id’␣==␣" ≪ curr_id ≪
      endl ≪ "***" ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

320. Look up *curr_id* in *keyword_map* and possibly *id_map*. [LDF 2006.10.19.]

```
⟨ Define yylex 302 ⟩ +≡
  ⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩
  }    /* else if (state ≡ COLLECTING_ID) */
  else
    if (state ≡ COLLECTING_INTEGER) {
      curr_integer_str += curr_char;
    }
    else if (state ≡ COLLECTING_FLOAT) {
      curr_float_str += curr_char;
    }
    else if (state ≡ COLLECTING_STRING) {
      curr_string += curr_char;
    }
  }    /* else if (isdigit(curr_char)) */
```

321. Escaped characters. '\. [LDF 2006.10.18.]

```
⟨ Define yylex 302 ⟩ +≡
  else if (curr_char ≡ '\\') {
```

322. *state* ≡ COLLECTING_ID. [LDF 2006.10.18.]

```
⟨ Define yylex 302 ⟩ +≡
  if (state ≡ COLLECTING_ID) {
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "***␣Finished␣collecting␣keyword.␣␣" ≪ "‘curr_id’␣==␣" ≪ curr_id ≪
      endl ≪ "***" ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

323. Look up *curr_id* in *keyword_map* and possibly *id_map*. [LDF 2006.10.19.]

```
⟨ Define yylex 302 ⟩ +≡
  ⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩
  }   /* if (state ≡ COLLECTING_ID) */
  else
    if (state ≡ COLLECTING_INTEGER) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣integer.␣␣" ≪ "‘curr_integer_str’␣==␣" ≪
          curr_integer_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→int_value = atoi(curr_integer_str.c_str());
      scanner_node→in_strm.unget();
      return INTEGER;
    }   /* else if (state ≡ COLLECTING_INTEGER) */
    else if (state ≡ COLLECTING_FLOAT) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣float.␣␣" ≪ "‘curr_float_str’␣==␣" ≪
          curr_float_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→float_value = atof(curr_float_str.c_str());
      scanner_node→in_strm.unget();
      return FLOAT;
    }   /* else if (state ≡ COLLECTING_FLOAT) */
```

324. *state* ≡ COLLECTING_STRING. [LDF 2006.10.18.]

```
⟨ Define yylex 302 ⟩ +≡
  if (state ≡ COLLECTING_STRING) {
    curr_char = scanner_node→in_strm.get();
    curr_string += curr_char;
    continue;
  }   /* if (state ≡ COLLECTING_STRING) */
```

325.

```
⟨ Define yylex 302 ⟩ +≡
  }   /* else if (curr_char ≡ '\\') */
```

326. Punctuation. [LDF 2006.10.31.]

Log

[LDF 2006.10.31.] Added this section.

327. `'='` (Equals sign). [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else if (curr_char ≡ '=') {
  if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "In␣‘yylex’:␣␣" ≪ "Got␣‘=’.␣␣Returning␣‘ASSIGN’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    return ASSIGN;
  }
  else if (state ≡ COLLECTING_STRING) {
    curr_string += curr_char;
  }
  else if (state ≡ COLLECTING_ID) {
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "***␣Finished␣collecting␣keyword.␣␣" ≪ "‘curr_id’␣==␣" ≪ curr_id ≪
      endl ≪ "***" ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

328. Look up *curr_id* in *keyword_map* and possibly *id_map*. [LDF 2006.10.19.]

```
⟨ Define yylex 302 ⟩ +≡
  ⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩
  }   /* else if (state ≡ COLLECTING_ID) */
  else
    if (state ≡ COLLECTING_INTEGER) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣integer.␣␣" ≪ "‘curr_integer_str’␣==␣" ≪
          curr_integer_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→int_value = atoi(curr_integer_str.c_str());
      scanner_node→in_strm.unget();
      return INTEGER;
    }   /* else if (state ≡ COLLECTING_INTEGER) */
    else if (state ≡ COLLECTING_FLOAT) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣float.␣␣" ≪ "‘curr_float_str’␣==␣" ≪
          curr_float_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→float_value = atof(curr_float_str.c_str());
      scanner_node→in_strm.unget();
      return FLOAT;
    }   /* else if (state ≡ COLLECTING_FLOAT) */
  }   /* else if (curr_char ≡ '=') */
```

329. `'+'` (Plus sign). [LDF 2006.11.13.]

Log

[LDF 2006.11.13.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '+') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣`yylex':␣␣" ≪ "Got␣`+'." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        next_char = scanner_node→in_strm.peek();
        if (next_char ≡ '=') {
          scanner_node→in_strm.get();
          return PLUS_ASSIGN;
        }
        else return PLUS;
      }    /* if (state ≡ NULL_STATE) */
      goto COMMON_PUNCTUATION;
    }    /* else if (curr_char ≡ '+') */
```

330. '&' (Ampersand). [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '&') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣`yylex':␣␣" ≪ "Got␣`&'." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        next_char = scanner_node→in_strm.peek();
        if (next_char ≡ '=') {
          scanner_node→in_strm.get();
          return AND_ASSIGN;
        }
        else if (next_char ≡ '!') {
          scanner_node→in_strm.get();
          return AND_NOT;
        }
        else return AND;
      }   /* if (state ≡ NULL_STATE) */
      goto COMMON_PUNCTUATION;
    }   /* else if (curr_char ≡ '&') */
```

331. '|' (Vertical line). [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '|') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’:␣␣" ≪ "Got␣‘|’." ≪ endl;
        cerr_mutex.Lock( );
        cerr ≪ temp_strm.str( );
        cerr_mutex.Unlock( );
        scanner_node→log_strm ≪ temp_strm.str( );
        temp_strm.str("");
#endif
        next_char = scanner_node→in_strm.peek( );
        if (next_char ≡ '=') {
          scanner_node→in_strm.get( );
          return OR_ASSIGN;
        }
        else if (next_char ≡ '!') {
          scanner_node→in_strm.get( );
          return OR_NOT;
        }
        else return OR;
      }    /* if (state ≡ NULL_STATE) */
      goto COMMON_PUNCTUATION;
    }    /* else if (curr_char ≡ '|') */
```

332. '^' (Circumflex accent). [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '^') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣`yylex':␣␣" ≪ "Got␣`^'." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        next_char = scanner_node→in_strm.peek();
        if (next_char ≡ '=') {
          scanner_node→in_strm.get();
          return XOR_ASSIGN;
        }
        else if (next_char ≡ '!') {
          scanner_node→in_strm.get();
          return XOR_NOT;
        }
        else return XOR;
      }    /* if (state ≡ NULL_STATE) */
      goto COMMON_PUNCTUATION;
    }    /* else if (curr_char ≡ '^') */
```

333. '!' (Exclamation mark). [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '!') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’:␣␣" ≪ "Got␣‘!’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        next_char = scanner_node→in_strm.peek();
        if (next_char ≡ '=') {
          scanner_node→in_strm.get();
          return NOT_ASSIGN;
        }
        else return NOT;
      }   /* if (state ≡ NULL_STATE) */
      else goto COMMON_PUNCTUATION;
    }   /* else if (curr_char ≡ '!') */
```

334. ':' (Colon). [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ ':') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’:␣␣" ≪ "Got␣‘:’.␣␣Returning␣‘COLON’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return COLON;
      }
      goto COMMON_PUNCTUATION;
    }   /* else if (curr_char ≡ ':') */
```

335. ';' (Semi-colon). [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ ';') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’:␣␣" ≪ "Got␣‘;’.␣␣Returning␣‘SEMI_COLON’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return SEMI_COLON;
      }
      goto COMMON_PUNCTUATION;
    }   /* else if (curr_char ≡ ';') */
```

336. ',' (Comma). [LDF 2006.12.06.]

Log

[LDF 2006.12.06.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ ',') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’,␣␣" ≪ "Got␣‘,’.␣␣Returning␣‘COMMA’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return COMMA;
      }
      goto COMMON_PUNCTUATION;
    }   /* else if (curr_char ≡ ',') */
```

337. '%' (Percent). [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '%') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’,␣␣" ≪ "Got␣‘%’.␣␣Returning␣‘PERCENT’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return PERCENT;
      }
      goto COMMON_PUNCTUATION;
    }    /* else if (curr_char ≡ '%') */
```

338. '-' (Hyphen). [LDF 2006.12.07.]

Log

[LDF 2006.12.07.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '-') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’,␣␣" ≪ "Got␣‘,’.␣␣Returning␣‘HYPHEN’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return HYPHEN;
      }
      goto COMMON_PUNCTUATION;
    }    /* else if (curr_char ≡ '-') */
```

339. '.' (Period). [LDF 2006.11.02.]

Please note: ⟨ Common code for punctuation 351 ⟩ cannot be used for period! It must be handled in a special way, because periods are used to distinguish integers and floats. [LDF 2006.11.14.]

Log

[LDF 2006.11.02.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else if (curr_char ≡ '.') {
  if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "In␣‘yylex’:␣␣" ≪ "Got␣‘.’.␣␣Returning␣‘PERIOD’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    return PERIOD;
  }
  else if (state ≡ COLLECTING_STRING) {
    curr_string += curr_char;
  }
  else if (state ≡ COLLECTING_ID) {
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "***␣Finished␣collecting␣id.␣␣" ≪ "‘curr_id’␣==␣" ≪ curr_id ≪ endl ≪
      "***" ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

340. Look up *curr_id* in *keyword_map* and possibly *id_map*. [LDF 2006.10.19.]

```
⟨ Define yylex 302 ⟩ +≡
  ⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩
  }   /* else if (state ≡ COLLECTING_ID) */
  else
    if (state ≡ COLLECTING_INTEGER) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ "***␣Read␣'.'␣while␣collecting␣integer.␣␣" ≪
          "Changing␣state␣to␣'COLLECTING_FLOAT'." ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      curr_float_str = curr_integer_str + curr_char;
      curr_integer_str = "";
      state = COLLECTING_FLOAT;
      continue;
    }   /* else if (state ≡ COLLECTING_INTEGER) */
    else if (state ≡ COLLECTING_FLOAT) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣float.␣␣" ≪ "'curr_float_str'␣==␣" ≪
          curr_float_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→float_value = atof(curr_float_str.c_str());
      scanner_node→in_strm.unget();
      return FLOAT;
    }   /* else if (state ≡ COLLECTING_FLOAT) */
  }   /* else if (curr_char ≡ '.') */
```

341. '@' (At-symbol). [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '@') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’@␣␣" ≪ "Got␣‘@’.␣␣Returning␣‘AT_SYMBOL’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return AT_SYMBOL;
      }
      goto COMMON_PUNCTUATION;
    }    /* else if (curr_char ≡ '@') */
```

342. '(' (Open parenthesis). [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '(') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’:␣␣" ≪ "Got␣‘(’.␣␣Returning␣‘OPEN_PARENTHESIS’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return OPEN_PARENTHESIS;
      }
      goto COMMON_PUNCTUATION;
    }    /* else if (curr_char ≡ '(') */
```

343. ')' (Close parenthesis). [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ ')') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’:␣␣" ≪ "Got␣‘)’.␣␣Returning␣‘CLOSE_PARENTHESIS’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return CLOSE_PARENTHESIS;
      }
      goto COMMON_PUNCTUATION;
    }   /* else if (curr_char ≡ ')') */
```

344. '[' (Open bracket). [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ '[') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣‘yylex’:␣␣" ≪ "Got␣‘[’.␣␣Returning␣‘OPEN_BRACKET’." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return OPEN_BRACKET;
      }
      goto COMMON_PUNCTUATION;
    }   /* else if (curr_char ≡ '[') */
```

345. ']' (Close bracket). [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Added this section.

```
⟨ Define yylex 302 ⟩ +≡
  else
    if (curr_char ≡ ']') {
      if (state ≡ NULL_STATE) {
#ifdef DEBUG_OUTPUT
        temp_strm ≪ "In␣'yylex':␣␣" ≪ "Got␣']'.␣␣Returning␣'CLOSE_BRACKET'." ≪ endl;
        cerr_mutex.Lock();
        cerr ≪ temp_strm.str();
        cerr_mutex.Unlock();
        scanner_node→log_strm ≪ temp_strm.str();
        temp_strm.str("");
#endif
        return CLOSE_BRACKET;
      }
      goto COMMON_PUNCTUATION;
    }   /* else if (curr_char ≡ ']') */
```

346. Space characters. [LDF 2006.10.18.]

```
⟨ Define yylex 302 ⟩ +≡
  else if (isspace(curr_char)) {
```

347. *state* ≡ NULL_STATE. Skip whitespace. [LDF 2006.10.18.]

```
⟨ Define yylex 302 ⟩ +≡
  if (state ≡ NULL_STATE) {
    continue;
  }   /* if (state ≡ NULL_STATE) */
  else if (state ≡ COLLECTING_ID) {
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "***␣Finished␣collecting␣keyword.␣␣" ≪ "'curr_id'␣==␣" ≪ curr_id ≪
      endl ≪ "***" ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

348. Look up *curr_id* in *keyword_map* and possibly *id_map*. [LDF 2006.10.19.]

```
⟨ Define yylex 302 ⟩ +≡
  ⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩
  }   /* else if (state ≡ COLLECTING_ID) */
```

349. *state* ≡ COLLECTING_STRING. [LDF 2006.10.18.]

⟨ Define *yylex* 302 ⟩ +≡

```
  else
    if (state ≡ COLLECTING_STRING) {
      curr_string += curr_char;
    }
    else if (state ≡ COLLECTING_INTEGER) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣integer.␣␣" ≪ "‘curr_integer_str’␣==␣" ≪
          curr_integer_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→int_value = atoi(curr_integer_str.c_str());
      scanner_node→in_strm.unget();
      return INTEGER;
    }    /* else if (state ≡ COLLECTING_INTEGER) */
    else if (state ≡ COLLECTING_FLOAT) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣float.␣␣" ≪ "‘curr_float_str’␣==␣" ≪
          curr_float_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      value→float_value = atof(curr_float_str.c_str());
      scanner_node→in_strm.unget();
      return FLOAT;
    }    /* else if (state ≡ COLLECTING_FLOAT) */
  }    /* else if (isspace(curr_char)) */
MAIN_LOOP_END: ; }    /* while (End of main loop) */
FINISH:
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "End␣of␣main␣loop." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

350. Close input file and exit successfully with return value 0. [LDF 2006.10.17.]

```
⟨ Define yylex 302 ⟩ +≡
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "Exiting␣‘yylex’." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
#undef DEBUG_OUTPUT
  return 0;
COMMON_PUNCTUATION: ⟨ Common code for punctuation 351 ⟩
  }    /* End of yylex definition. */
```

351. Common code for punctuation. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this declaration.

```
⟨ Common code for punctuation 351 ⟩ ≡
  if (state ≡ COLLECTING_STRING) {
    curr_string += curr_char;
    goto MAIN_LOOP_END;
  }
  else if (state ≡ COLLECTING_ID) {
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "***␣Finished␣collecting␣keyword.␣␣" ≪ "‘curr_id’␣==␣" ≪ curr_id ≪
      endl ≪ "***" ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

See also section 352.
This code is cited in section 339.
This code is used in section 350.

352. Look up *curr_id* in *keyword_map* and possibly *id_map*. [LDF 2006.10.19.]

```
⟨ Common code for punctuation 351 ⟩ +≡
  ⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩
  }    /* else if (state ≡ COLLECTING_ID) */
  else
    if (state ≡ COLLECTING_INTEGER) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣integer.␣␣" ≪ "‘curr_integer_str’␣==␣" ≪
          curr_integer_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock( );
      cerr ≪ temp_strm.str( );
      cerr_mutex.Unlock( );
      scanner_node→log_strm ≪ temp_strm.str( );
      temp_strm.str("");
#endif
      value→int_value = atoi(curr_integer_str.c_str( ));
      scanner_node→in_strm.unget( );
      return INTEGER;
    }    /* else if (state ≡ COLLECTING_INTEGER) */
    else if (state ≡ COLLECTING_FLOAT) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ endl ≪ "***␣Finished␣collecting␣float.␣␣" ≪ "‘curr_float_str’␣==␣" ≪
          curr_float_str ≪ "␣␣***" ≪ endl;
      cerr_mutex.Lock( );
      cerr ≪ temp_strm.str( );
      cerr_mutex.Unlock( );
      scanner_node→log_strm ≪ temp_strm.str( );
      temp_strm.str("");
#endif
      value→float_value = atof(curr_float_str.c_str( ));
      scanner_node→in_strm.unget( );
      return FLOAT;
    }    /* else if (state ≡ COLLECTING_FLOAT) */
```

353. Look up *curr_id*. Look up *curr_id* in *keyword_map* and possibly *id_map*. [LDF 2006.11.01.]

A consequence of looking up *curr_id* in *keyword_map* first, and then looking it up in *id_map*, is that it's not possible for variables declared by users to shadow keywords. I could change this, if it were desirable. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this section. It's included in *yylex*.

```
⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩ ≡
  map⟨string, Keyword_Type *⟩::iterator keyword_iter = keyword_map.find(curr_id); if
      (keyword_iter ≡ keyword_map.end( )) {
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "In␣‘yylex’:␣␣" ≪ "‘" ≪ curr_id ≪ "’␣not␣found␣in␣‘keyword_map’." ≪
      endl ≪ "Will␣search␣in␣‘scanner_node->id_map’." ≪ endl;
  cerr_mutex.Lock( );
  cerr ≪ temp_strm.str( );
  cerr_mutex.Unlock( );
```

```
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
  map⟨string, Id_Type *⟩::iterator id_iter = scanner_node→id_map.find(curr_id);
```

See also sections 354 and 355.

This code is used in sections 310, 320, 323, 328, 340, 348, and 352.

354. *curr_id* not found in *scanner_node→id_map*. [LDF 2006.11.01.]

```
⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩ +≡
  if (id_iter ≡ scanner_node→id_map.end()) {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "In␣‘yylex’:␣␣‘" ≪ curr_id ≪ "’␣not␣found␣in␣" ≪
        "‘scanner_node->id_map’.␣␣Returning␣‘VARIABLE_TEXT_SEGMENT’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str();
#endif
    strcpy(value→string_value, curr_id.c_str());
    curr_id = "";
    scanner_node→in_strm.unget();
    return VARIABLE_TEXT_SEGMENT;
  }   /* if (id_iter ≡ scanner_node→id_map.end()) (curr_id not found in scanner_node→id_map) */
```

355. *curr_id* found in *scanner_node→id_map*. [LDF 2006.11.01.]

```
⟨ Look up curr_id in keyword_map and possibly id_map 353 ⟩ +≡
  else    /* id_iter ≠ scanner_node→id_map.end() */
  {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "In␣‘yylex’:␣␣‘" ≪ curr_id ≪ "’␣found␣in␣" ≪ "‘scanner_node->id_map’." ≪
        endl ≪ "Pushing␣‘Token_Type’␣object␣onto␣‘scanner_node->token_stack’␣" ≪
        "and␣returning␣‘VARIABLE’␣(" ≪ VARIABLE ≪ ")." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str();
#endif
    Token_Type curr_token;
    curr_token.value.pointer_value = scanner_node→id_map[curr_id];
    curr_token.type = scanner_node→id_map[curr_id]→type;
    scanner_node→token_stack.push(curr_token);
    scanner_node→in_strm.unget();
    curr_id = "";
    value→int_value = 0;
    return VARIABLE;
  }    /* else (id_iter ≠ scanner_node→id_map.end()) */
  }    /* if (keyword_iter ≡ keyword_map.end()) (curr_id not found in keyword_map) */
  else    /* keyword_iter ≠ keyword_map.end() (Keyword found) */
  {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "Keyword␣‘" ≪ curr_id ≪ "’␣found." ≪ endl ≪
        "Exiting␣function␣successfully␣with␣return␣value␣" ≪ keyword_iter→second→value_name ≪
        "␣(" ≪ keyword_iter→second→value ≪ ")" ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    scanner_node→in_strm.unget();
    return keyword_iter→second→value;
  }    /* else (keyword_iter ≠ keyword_map.end()) (Keyword found) */
```

356. Putting the scanner together.

357. This is what's compiled.

```
⟨ Include files 10 ⟩
⟨ scanner.web 245 ⟩
using namespace std;
⟨ Forward declarations 13 ⟩
⟨ Declare namespace Scan_Parse 251 ⟩
⟨ Define Scan_Parse functions 265 ⟩
⟨ Define yylex 302 ⟩
```

358. This is what's written to `scanner.h`.

⟨ `scanner.hh` 358 ⟩ ≡
⟨ Preprocessor macro calls 19 ⟩
using namespace std;
⟨ Forward declarations 13 ⟩
⟨ **extern** declaration of namespace **Scan_Parse** 252 ⟩
⟨ Declare *yylex* 301 ⟩

359. Parser (`parser.w`). [LDF 2006.10.20.]

Log

[2006.10.20.] Added this file.

[LDF 2006.10.31.] Renamed this file. Old name: `parser.web`. New name: `parser.w`.

⟨ `parser.w` 359 ⟩ ≡
static char *parser_id_string*[ ] = `"$Id:␣parser.w,v␣1.5␣2007/01/02␣12:38:44␣lfinsto1␣Exp␣$"`;
This code is cited in sections 6, 7, 8, and 644.
This code is used in section 642.

360. Include files. [LDF 2006.10.20.]

⟨ Include files 10 ⟩ +≡
```
#include "localldf.h"
#ifdef NON_WIN_LDF
#include "nonwin.h"
#else
#include "stdafx.h"
#endif
#include "parser.hxx"
#include "dttmtype.h"
#include "scnrtype.h"
#include "scanner.h"
#include "querytyp.h"
#include "dtsrctyp.h"
#include "idtype.h"
```

361. Declare *yylex*. [LDF 2006.10.20.]

⟨ Declare *yylex* 301 ⟩ +≡
int *yylex*(`YYSTYPE` ∗ *lvalp*, **void** ∗*parameter*);

362. Declare *yyerror*. [LDF 2006.10.20.]

⟨ Declare *yyerror* 362 ⟩ ≡
void *yyerror*(**const char** ∗*s*);
This code is used in section 643.

363. union declaration for `YYSTYPE`.

⟨ **union** declaration for `YYSTYPE` 363 ⟩ ≡
`%union␣{`
int *int_value*;
float *float_value*;

char *string_value*[2048];
void **pointer_value*; }

This code is used in section 643.

364. Bison declarations. [LDF 2006.10.20.]

⟨ Bison declarations 364 ⟩ ≡

```
%pure_parser %debug
```

This code is used in section 643.

365. Token and Type Declarations. [LDF 2006.10.20.]

366. Punctuation. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

[LDF 2006.12.07.] Added token declaration for HYPHEN.

[LDF 2006.12.08.] Added token declaration for UNDERLINE.

[LDF 2006.12.12.] Added token declaration for AT_SYMBOL.

[LDF 2006.12.14.] Added token declaration for PERCENT.

⟨ Token and type declarations 366 ⟩ ≡

```
%token <int_value> AT_SYMBOL
%token <int_value> COMMA
%token <int_value> COLON
%token <int_value> HYPHEN
%token <int_value> UNDERLINE
%token <int_value> PERCENT
%token <int_value> PERIOD
%token <int_value> SEMI_COLON
%token <int_value> OPEN_PARENTHESIS
%token <int_value> CLOSE_PARENTHESIS
%token <int_value> OPEN_BRACKET
%token <int_value> CLOSE_BRACKET
```

See also sections 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 382, 383, 384, and 385.

This code is used in section 643.

367. Assignments. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section with the declarations of ASSIGN, PLUS_ASSIGN, MINUS_ASSIGN, TIMES_ASSIGN, and DIVIDE_ASSIGN.

[LDF 2006.11.14.] Added the declarations of AND_ASSIGN, OR_ASSIGN, XOR_ASSIGN, and NOT_ASSIGN.

[LDF 2006.12.11.] Added the declarations of LIKE and FREETEXT.

[LDF 2006.12.12.] Added the declaration of CONTAINS.

⟨ Token and type declarations 366 ⟩ +≡

```
%token␣<int_value>␣ASSIGN
%token␣<int_value>␣PLUS_ASSIGN
%token␣<int_value>␣MINUS_ASSIGN
%token␣<int_value>␣TIMES_ASSIGN
%token␣<int_value>␣DIVIDE_ASSIGN
%token␣<int_value>␣AND_ASSIGN
%token␣<int_value>␣OR_ASSIGN
%token␣<int_value>␣XOR_ASSIGN
%token␣<int_value>␣NOT_ASSIGN
%token␣<int_value>␣CONTAINS
%token␣<int_value>␣FREETEXT
%token␣<int_value>␣LIKE
```

368. Arithmetical Operations. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this section with the token declarations of PLUS, MINUS, TIMES, and DIVIDE.

⟨ Token and type declarations 366 ⟩ +≡

```
%token␣<int_value>␣PLUS
%token␣<int_value>␣MINUS
%token␣<int_value>␣TIMES
%token␣<int_value>␣DIVIDE
```

369. Types with values. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

⟨ Token and type declarations 366 ⟩ +≡

```
%token␣<string_value>␣STRING
%token␣<int_value>␣␣␣␣INTEGER
%token␣<float_value>␣␣FLOAT
```

370. Control. [LDF 2006.10.20.]

⟨ Token and type declarations 366 ⟩ +≡

```
%token <int_value> START
%token <int_value> END
%token <int_value> TERMINATE
```

371. Declarators. [LDF 2006.12.15.]

Log

[LDF 2006.11.16.] Added token declaration for STRING_DECLARATOR.

[LDF 2006.12.08.] Added token declaration for DATASOURCE_DECLARATOR.

[LDF 2006.12.15.] Added token declaration for DATETIME_DECLARATOR.

⟨ Token and type declarations 366 ⟩ +≡

```
%token <int_value> DATASOURCE_DECLARATOR
%token <int_value> DATETIME_DECLARATOR
%token <int_value> STRING_DECLARATOR
%token <int_value> QUERY_DECLARATOR
```

372. Queries. [LDF 2006.10.20.]

Log

[LDF 2006.10.31.] Added token declarations for DATABASE and SERVER.

⟨ Token and type declarations 366 ⟩ +≡

```
%token <int_value> LOCAL
%token <int_value> REMOTE
%token <int_value> DATABASE
%token <int_value> SERVER
```

373. Datasources. [LDF 2006.12.08.]

Log

[2006.12.08.] Added this section with the token declarations for DBT, GBV_GVK, TIMMS, and DATASOURCE_FILE. LOCAL, DATABASE, and SERVER have already been defined, as they are used for queries.

⟨ Token and type declarations 366 ⟩ +≡

```
%token <int_value> DBT
%token <int_value> GBV_GVK
%token <int_value> TIMMS
%token <int_value> DATASOURCE_FILE
```

374. Datetimes. [LDF 2006.12.15.]

Log

[2006.12.15.] Added this section with the token declarations for DAY, MONTH, YEAR, HOUR, MINUTE, and SECOND.

[LDF 2006.12.18.] Added token declaration for YEAR_RANGE_BEGIN and YEAR_RANGE_END.

⟨ Token and type declarations 366 ⟩ +≡

```
%token <int_value>   YEAR_RANGE_BEGIN
%token <int_value>   YEAR_RANGE_END
%token <int_value>   YEAR
%token <int_value>   MONTH
%token <int_value>   DAY
%token <int_value>   HOUR
%token <int_value>   MINUTE
%token <int_value>   SECOND
```

375. Predicates and Control Structures. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this section with the token declarations for IF, ELSE, ELIF, FI, FOR, DO, WHILE, AND, OR, XOR, NOT, AND_NOT, OR_NOT, and XOR_NOT.

⟨ Token and type declarations 366 ⟩ +≡

```
%token <int_value> IF
%token <int_value> ELSE
%token <int_value> ELIF
%token <int_value> FI
%token <int_value> FOR
%token <int_value> DO
%token <int_value> WHILE
%token <int_value> AND
%token <int_value> OR
%token <int_value> XOR
%token <int_value> NOT
%token <int_value> AND_NOT
%token <int_value> OR_NOT
%token <int_value> XOR_NOT
```

376. Database tables (Field designators). [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section with token declarations for `AUTHOR`, `CONTRIBUTOR`, `TITLE`, and `SUBJECT`.

[LDF 2006.12.05.] Added the token declaration for `CREATOR`.

[LDF 2006.12.12.] Added the token declaration for `MAIN_CANONICAL_TITLE`.

[LDF 2006.12.12.] Added token declarations for additional tables from the OAI database (dc_test).

[LDF 2006.12.12.] Added token declarations for additional tables from the PICA database (PICA_DB).

⟨ Token and type declarations 366 ⟩ +≡

```
%token␣<int_value>␣ACCESS_NUMBER
%token␣<int_value>␣AUTHOR
%token␣<int_value>␣BIBLIOGRAPHIC_TYPE
%token␣<int_value>␣CALL_NUMBER
%token␣<int_value>␣CLASSIFICATION
%token␣<int_value>␣COMPANY
%token␣<int_value>␣CONTENT_SUMMARY
%token␣<int_value>␣CONTRIBUTOR
%token␣<int_value>␣CREATOR
%token␣<int_value>␣DATABASE_PROVIDER
%token␣<int_value>␣DESCRIPTION
%token␣<int_value>␣EXEMPLAR_PRODUCTION_NUMBER
%token␣<int_value>␣IDENTIFIER
%token␣<int_value>␣INSTITUTION
%token␣<int_value>␣LANGUAGE
%token␣<int_value>␣MAIN_CANONICAL_TITLE
%token␣<int_value>␣PERMUTATION_PATTERN
%token␣<int_value>␣PERSON
%token␣<int_value>␣PHYSICAL_DESCRIPTION
%token␣<int_value>␣PUBLISHER
%token␣<int_value>␣RECORD
%token␣<int_value>␣REMOTE_ACCESS
%token␣<int_value>␣RIGHTS
%token␣<int_value>␣SOURCE
%token␣<int_value>␣SUBJECT
%token␣<int_value>␣SUPERORDINATE_ENTITIES
%token␣<int_value>␣TITLE
%token␣<int_value>␣TYPE
```

377. Database table columns (field qualifiers). [LDF 2006.12.13.]

Log

[2006.12.13.] Added this section.

⟨ Token and type declarations 366 ⟩ +≡

378. Generic columns. [LDF 2006.12.13.]

Log

[2006.12.13.] Added this section.

⟨ Token and type declarations 366 ⟩ +≡

```
%token␣&lt;int_value&gt;␣ID
```

379. Names. [LDF 2006.12.07.]

Log

[LDF 2006.12.07.] Added this section with token declarations for `GIVEN_NAME`, `PREFIX`, and `SURNAME`.

[LDF 2006.12.07.] Added token declarations for `AUTHOR_GIVEN_NAME`, `AUTHOR_PREFIX`, `AUTHOR_SURNAME`, `CONTRIBUTOR_GIVEN_NAME`, `CONTRIBUTOR_PREFIX`, and `CONTRIBUTOR_SURNAME`.

⟨ Token and type declarations 366 ⟩ +≡

```
%token␣<int_value>␣GIVEN_NAME
%token␣<int_value>␣PREFIX
%token␣<int_value>␣SURNAME
%token␣<int_value>␣AUTHOR_GIVEN_NAME
%token␣<int_value>␣AUTHOR_PREFIX
%token␣<int_value>␣AUTHOR_SURNAME
%token␣<int_value>␣CONTRIBUTOR_GIVEN_NAME
%token␣<int_value>␣CONTRIBUTOR_PREFIX
%token␣<int_value>␣CONTRIBUTOR_SURNAME
```

380. Columns of individual tables. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

---

381. Pica. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

---

382. Records. [LDF 2006.12.14.]
`ID` in "Generic" section. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

⟨ Token and type declarations 366 ⟩ +≡

```
%token␣&lt;int_value&gt;␣ELN_ORIGINAL_ENTRY
%token␣&lt;int_value&gt;␣ELN_MOST_RECENT_CHANGE
%token␣&lt;int_value&gt;␣ELN_STATUS_CHANGE
%token␣&lt;int_value&gt;␣IDENTIFICATION_NUMBER
%token␣&lt;int_value&gt;␣DATE_ORIGINAL_ENTRY
%token␣&lt;int_value&gt;␣DATE_MOST_RECENT_CHANGE
%token␣&lt;int_value&gt;␣DATE_STATUS_CHANGE
%token␣&lt;int_value&gt;␣SOURCE_ID
%token␣&lt;int_value&gt;␣YEAR_APPEARANCE_BEGIN
%token␣&lt;int_value&gt;␣YEAR_APPEARANCE_END
%token␣&lt;int_value&gt;␣YEAR_APPEARANCE_RAK_WB
%token␣&lt;int_value&gt;␣YEAR_APPEARANCE_ORIGINAL
```

383. Variables. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this section.

⟨ Token and type declarations 366 ⟩ +≡

```
%token␣&lt;int_value&gt;␣VARIABLE
%token␣&lt;string_value&gt;␣VARIABLE_TEXT_SEGMENT
```

384. Data Types. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this section with the token declaration of `QUERY_TYPE`.

[LDF 2006.11.03.] Added the token declaration of `NULL_TYPE`.

[LDF 2006.11.16.] Added the token declaration of `STRING_TYPE`.

[LDF 2006.12.08.] Added the token declaration of `DATASOURCE_TYPE`.

[LDF 2006.12.15.] Added the token declaration of `DATETIME_TYPE`.

⟨ Token and type declarations 366 ⟩ +≡

```
%token␣&lt;pointer_value&gt;␣NULL_TYPE
%token␣&lt;pointer_value&gt;␣DATASOURCE_TYPE
%token␣&lt;pointer_value&gt;␣DATETIME_TYPE
%token␣&lt;pointer_value&gt;␣QUERY_TYPE
%token␣&lt;pointer_value&gt;␣STRING_TYPE
```

385. Commands. [LDF 2006.11.13.]

Log

[LDF 2006.11.13.] Added this section with the token declarations of `SHOW` and `CLEAR`.

[LDF 2006.11.13.] Added the token declarations of `MESSAGE`, `ERRMESSAGE`, and `PAUSE`.

[LDF 2006.11.16.] Added the token declarations of `TEX`, `SQL`, `OAI`, and `PICA`.

[LDF 2006.11.23.] Added the token declaration of `OUTPUT`.

⟨ Token and type declarations 366 ⟩ +≡

```
%token <int_value> CLEAR
%token <int_value> ERRMESSAGE
%token <int_value> MESSAGE
%token <int_value> PAUSE
%token <int_value> SHOW
%token <int_value> TEX
%token <int_value> SQL
%token <int_value> OAI
%token <int_value> PICA
%token <int_value> OUTPUT
```

386. Parser rules. [LDF 2006.10.20.]

387. Program. [LDF 2006.10.31.]

388. ⟨program⟩ ⟶ ⟨statement list⟩ `TERMINATE`. [LDF 2006.10.31.]

⟨ Parser rules 388 ⟩ ≡

```
program: statement_list TERMINATE
{
  Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
  temp_strm << endl << "Rule 'statement_list: TERMINATE'." << endl;
  cerr << temp_strm.str();
  scanner_node->log_strm << temp_strm.str();
  temp_strm.str("");
  return 0;
}
;
```

See also sections 390, 391, 394, 395, 396, 403, 404, 405, 406, 409, 410, 411, 412, 414, 416, 418, 420, 424, 426, 427, 428, 430, 431, 432, 433, 435, 436, 438, 439, 441, 442, 444, 445, 449, 450, 452, 453, 454, 455, 457, 458, 460, 461, 462, 463, 464, 465, 466, 467, 469, 470, 472, 473, 474, 475, 477, 478, 479, 480, 481, 482, 484, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 515, 516, 519, 521, 522, 523, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 539, 540, 541, 542, 543, 545, 546, 547, 548, 549, 550, 551, 553, 554, 555, 557, 558, 559, 560, 561, 562, 563, 564, 569, 570, 571, 572, 573, 574, 575, 576, 577, 581, 582, 583, 584, 585, 587, 588, 590, 591, 593, 594, 596, 597, 598, 599, 600, 601, 605, 606, 608, 610, 612, 615, 616, 617, 619, 621, 623, 627, 628, 629, 631, 633, 635, 637, 638, 639, and 640.

This code is used in section 643.

389. Statement list. [LDF 2006.10.31.]

390. ⟨statement list⟩ ⟶ EMPTY. [LDF 2006.10.31.]

⟨ Parser rules 388 ⟩ +≡

```
statement_list: /* empty */ ;
```

391. ⟨statement list⟩ ⟶ ⟨statement list⟩ ⟨statement⟩. [LDF 2006.10.31.]

⟨ Parser rules 388 ⟩ +≡

```
statement_list: statement_list statement ;
```

392. Statement. [LDF 2006.10.31.]

393. ⟨statement⟩ ⟶ ⟨group statement⟩ SEMI_COLON. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this rule.

[LDF 2006.11.13.] Commented-out this rule.

⟨ Garbage 393 ⟩ ≡

```
{
  Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
  temp_strm ≪ endl ≪ "Rule ‘statement: group_statement SEMI_COLON’." ≪ endl;
  cerr ≪ temp_strm.str();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
}
;
```

This code is used in section 644.

394. ⟨statement⟩ ⟶ ⟨declaration⟩ SEMI_COLON. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
statement: declaration SEMI_COLON
{
  Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
  temp_strm ≪ endl ≪ "Rule ‘statement: declaration SEMI_COLON’." ≪ endl;
  cerr ≪ temp_strm.str();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
}
;
```

395. ⟨statement⟩ ⟶ ⟨assignment⟩ `SEMI_COLON`. [LDF 2006.10.31.]

⟨ Parser rules 388 ⟩ +≡

```
statement:␣assignment␣SEMI_COLON
{
  Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
  temp_strm ≪ endl ≪ "Rule␣‘statement:␣assignment␣SEMI_COLON’." ≪ endl;
  cerr ≪ temp_strm.str();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
}
;
```

396. ⟨statement⟩ ⟶ ⟨command⟩ `SEMI_COLON`. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
statement:␣command␣SEMI_COLON
{
  Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
  temp_strm ≪ endl ≪ "Rule␣‘statement:␣command␣SEMI_COLON’." ≪ endl;
  cerr ≪ temp_strm.str();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
}
;
```

397. Group Statement (`grpstmnt.w`). [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

[LDF 2006.11.13.] Removed the rules from this file and commented-out the type declaration of *group_statement*.

398.

⟨ `grpstmnt.w` 398 ⟩ ≡

```
static char grpstmnt_id_string[] = "$Id:␣grpstmnt.w,v␣1.3␣2007/01/02␣12:38:44␣lfinsto1␣Exp␣$";
```

This code is cited in sections 7 and 8.

This code is used in section 642.

399.

⟨ Type declarations for non-terminal symbols 399 ⟩ ≡

See also sections 402, 407, 408, 413, 415, 417, 419, 423, 425, 429, 434, 437, 440, 443, 447, 448, 451, 456, 459, 468, 483, 485, 514, 517, 538, 544, 552, 556, 566, 568, 580, 586, 589, 592, 595, 604, 607, 609, 611, 614, 618, 620, 622, 626, 630, 632, 634, and 636.

This code is used in section 643.

400. Declarations (declrtns.w). [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Added this file.

401.

⟨ declrtns.w 401 ⟩ ≡

```
static char declrtns_id_string[] = "$Id: declrtns.w,v 1.3 2007/01/02 12:36:50 lfinsto1 Exp $";
```

This code is cited in sections 7 and 8.

This code is used in section 642.

402.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <int_value> declaration
```

403. ⟨declaration⟩ ⟶ ⟨query declaration⟩. [LDF 2006.11.01.]

⟨ Parser rules 388 ⟩ +≡

```
declaration: query_declaration
{
  Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
  temp_strm << endl << "Rule 'declaration: query_declaration'." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node->log_strm << temp_strm.str();
  temp_strm.str("");
}
;
```

404. ⟨declaration⟩ ⟶ ⟨string declaration⟩**.** [LDF 2006.11.16.]

Log

[2006.11.16.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
declaration: string_declaration
{
  Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
  temp_strm << endl << "Rule 'declaration: string_declaration'." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node->log_strm << temp_strm.str();
  temp_strm.str("");
}
;
```

405. ⟨declaration⟩ ⟶ ⟨datasource declaration⟩**.** [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
declaration: datasource_declaration
{
  Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
  temp_strm << endl << "Rule 'declaration: datasource_declaration'." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node->log_strm << temp_strm.str();
  temp_strm.str("");
}
;
```

406. ⟨declaration⟩ ⟶ ⟨datetime declaration⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
declaration: datetime_declaration
{
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'declaration: datetime_declaration'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
}
;
```

407. ⟨variable declaration segment list⟩. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <string_value> variable_declaration_segment_list
```

408. ⟨subscript placeholder⟩. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <int_value> subscript_placeholder
```

409. ⟨variable declaration segment list⟩ ⟶ `VARIABLE_TEXT_SEGMENT`. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
variable_declaration_segment_list:␣VARIABLE_TEXT_SEGMENT
{
  Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
  strcpy($$, $1);
  temp_strm ≪ endl ≪ "Rule␣‘variable_declaration_segment_list:␣VARIABLE_TEXT_SEGMENT’." ≪
      endl ≪ "‘$$’␣==␣" ≪ $$ ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
}
;
```

410. ⟨variable declaration segment list⟩ ⟶ **(etc.).** ⟨variable declaration segment list⟩ ⟶ ⟨variable declaration segment list⟩ `VARIABLE_TEXT_SEGMENT`. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
variable_declaration_segment_list:␣variable_declaration_segment_list␣ VARIABLE_TEXT_SEGMEN
{
  Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
  strcat($$, $2);
  temp_strm ≪ endl ≪ "Rule␣‘variable_declaration_segment_list:␣" ≪
      "variable_declaration_segment_list␣" ≪ "VARIABLE_TEXT_SEGMENT’." ≪ endl ≪
      "‘$$’␣==␣" ≪ $$ ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
}
;
```

411. ⟨variable declaration segment list⟩ ⟶ **(etc.).** ⟨variable declaration segment list⟩ ⟶ ⟨variable declaration segment list⟩ ⟨subscript placeholder⟩. [LDF 2006.11.01.]
The "Yen" symbol "¥" (#A5) is added to the semantic value of the
⟨variable declaration segment list⟩ as a placeholder. This string will be used as the name of a top-level **Id_Type** object in *scanner_node→id_map*. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
variable_declaration_segment_list:␣variable_declaration_segment_list␣subscript_placeholde
{
  Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
  strcat($$,"¥");
  temp_strm ≪ endl ≪ "Rule␣‘variable_declaration_segment_list:␣variable_declarati\
      on_segment_list␣" ≪ "subscript_placeholder’." ≪ endl ≪ "‘$$’␣==␣" ≪ $$ ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
}
;
```

412. ⟨subscript placeholder⟩ ⟶ `OPEN_BRACKET CLOSE_BRACKET`. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
subscript_placeholder:␣OPEN_BRACKET␣CLOSE_BRACKET
{
  Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
  temp_strm ≪ endl ≪ "Rule␣‘subscript_placeholder:␣OPEN_BRACKET␣CLOSE_BRACKET’." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
}
;
```

413. ⟨query declaration⟩**.** [LDF 2006.11.01.]

Log

[2006.11.01.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

`%type␣<int_value>␣query_declaration`

414. ⟨query declaration⟩ ⟶ **(etc.).** ⟨query declaration⟩ ⟶ `QUERY_DECLARATOR` ⟨variable declaration segment list⟩. [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Added this rule.

[LDF 2006.11.02.] No longer setting `$$` to the return value of **Scan_Parse** :: *declare_variable_func*, because I've changed its return value from **int** to **Id_Node**.

⟨ Parser rules 388 ⟩ +≡

```
query_declaration:␣QUERY_DECLARATOR␣variable_declaration_segment_list
{
  Scan_Parse::declare_variable_func(parameter, QUERY_TYPE, $2);
  $$ = 0;
}
;
```

415. ⟨datasource declaration⟩**.** [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

`%type␣<int_value>␣datasource_declaration`

416. ⟨datasource declaration⟩ ⟶ **(etc.).** ⟨datasource declaration⟩ ⟶ `DATASOURCE_DECLARATOR` ⟨variable declaration segment list⟩. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datasource_declaration:␣DATASOURCE_DECLARATOR␣variable_declaration_segment_list
{
  Scan_Parse::declare_variable_func(parameter, DATASOURCE_TYPE, $2);
  $$ = 0;
}
;
```

417. ⟨string declaration⟩**.** [LDF 2006.11.01.]

Log

[2006.11.01.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <int_value> string_declaration
```

418. ⟨string declaration⟩ ⟶ **(etc.).** ⟨string declaration⟩ ⟶ `STRING_DECLARATOR` ⟨variable declaration segment list⟩. [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Added this rule.

[LDF 2006.11.02.] No longer setting `$$` to the return value of **Scan_Parse** :: *declare_variable_func*, because I've changed its return value from **int** to **Id_Node**.

⟨ Parser rules 388 ⟩ +≡

```
string_declaration: STRING_DECLARATOR variable_declaration_segment_list
{
  Scan_Parse::declare_variable_func(parameter, STRING_TYPE, $2);
  $$ = 0;
}
;
```

419. ⟨datetime declaration⟩**.** [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <int_value> datetime_declaration
```

420. ⟨datetime declaration⟩ ⟶ **(etc.).** ⟨datetime declaration⟩ ⟶ `DATETIME_DECLARATOR` ⟨variable declaration segment list⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datetime_declaration: DATETIME_DECLARATOR variable_declaration_segment_list
{
  Scan_Parse::declare_variable_func(parameter, DATETIME_TYPE, $2);
  $$ = 0;
}
;
```

421. Variables (`variabls.w`). [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this file.

422.

⟨ `variabls.w` 422 ⟩ ≡

```
static char variabls_id_string[] = "$Id: variabls.w,v 1.3 2007/01/02 12:38:44 lfinsto1 Exp $";
```

This code is cited in sections 7 and 8.

This code is used in section 642.

423. ⟨variable name⟩. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> variable_name
```

424. ⟨variable name⟩ ⟶ ⟨variable segment list⟩. [LDF 2006.11.02.]

⟨ Parser rules 388 ⟩ +≡

```
variable_name: variable_segment_list
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Rule 'variable_name: variable_segment_list'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
#if 0     /* 1 */
    Token_Type curr_token;
    curr_token.type = yychar;
    curr_token.value = yylval;
#endif
    scanner_node->token_stack.push(Token_Type(yychar, yylval));
    $$ = Scan_Parse::variable_func(parameter, $1);
    yyclearin;
#undef DEBUG_OUTPUT
  }
  ;
```

425. ⟨variable segment list⟩. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <string_value> variable_segment_list
```

426. ⟨variable segment list⟩ ⟶ `VARIABLE_TEXT_SEGMENT`. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  variable_segment_list:␣VARIABLE_TEXT_SEGMENT
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    scanner_node→float_vector.clear();
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "Rule␣‘variable_segment_list:␣VARIABLE_TEXT_SEGMENT’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    strcpy($$, $1);
#undef DEBUG_OUTPUT
  }
  ;
```

427. ⟨variable segment list⟩ ⟶ **(etc.).** ⟨variable segment list⟩ ⟶ ⟨variable segment list⟩ VARIABLE_TEXT_SEGMENT.
[LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  variable_segment_list: variable_segment_list VARIABLE_TEXT_SEGMENT
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule ‘variable_segment_list: variable_segment_list " ≪
        "VARIABLE_TEXT_SEGMENT’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    strcpy($$, $1);
    strcat($$, $2);
#undef DEBUG_OUTPUT
  }
  ;
```

428. ⟨variable segment list⟩ ⟶ **(etc.).** ⟨variable segment list⟩ ⟶ ⟨variable segment list⟩ ⟨subscript⟩. [LDF 2006.11.03.]

Log

[LDF 2006.11.03.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  variable_segment_list: variable_segment_list subscript
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    scanner_node->float_vector.push_back($2);
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Rule `variable_segment_list: variable_segment_list " <<
        "subscript'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    strcat($$, "¥");
#undef DEBUG_OUTPUT
  }
  ;
```

429. ⟨subscript⟩. [LDF 2006.11.03.]

Log

[LDF 2006.11.03.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
  %type <float_value> subscript
```

430. ⟨subscript⟩ ⟶ `INTEGER`. [LDF 2006.11.03.]

Log

[LDF 2006.11.03.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  subscript:␣INTEGER
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm << endl << "Rule␣‘subscript:␣INTEGER’." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = static_cast⟨float⟩($1);
#undef DEBUG_OUTPUT
  }
  ;
```

431. ⟨subscript⟩ ⟶ `FLOAT`. [LDF 2006.11.03.]

Log

[LDF 2006.11.03.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  subscript: FLOAT
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `subscript: FLOAT'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $1;
#undef DEBUG_OUTPUT
  }
  ;
```

432. ⟨subscript⟩ ⟶ OPEN_BRACKET INTEGER CLOSE_BRACKET. [LDF 2006.11.03.]

Log

[LDF 2006.11.03.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  subscript: OPEN_BRACKET INTEGER CLOSE_BRACKET
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm << endl << "Rule 'subscript: OPEN_BRACKET INTEGER CLOSE_BRACKET'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = static_cast⟨float⟩($2);
#undef DEBUG_OUTPUT
  }
  ;
```

433. ⟨subscript⟩ ⟶ OPEN_BRACKET FLOAT CLOSE_BRACKET. [LDF 2006.11.03.]

Log

[LDF 2006.11.03.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  subscript: OPEN_BRACKET FLOAT CLOSE_BRACKET
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `subscript: OPEN_BRACKET FLOAT CLOSE_BRACKET'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $2;
#undef DEBUG_OUTPUT
  }
  ;
```

434. ⟨query variable⟩. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this type declaration.

```
⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
  %type <pointer_value> query_variable
```

435. ⟨query variable⟩ ⟶ `VARIABLE QUERY TYPE`. [LDF 2006.11.02.]
The semantic value of the `QUERY_TYPE` token is the **Id_Type** object associated with the name specified by the semantic value of the `VARIABLE` token, or 0, if there is no **Id_Type** object. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  query_variable:␣VARIABLE␣QUERY_TYPE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘query_variable:␣VARIABLE␣QUERY_TYPE’." ≪ endl;
    if ( $2 ≡ 0) {
      temp_strm ≪ "The␣semantic␣value␣of␣‘QUERY_TYPE’␣==␣0.␣␣Not␣showing." ≪ endl;
    }
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
    if ( $2 ≠ 0) {
      static_cast⟨Id_Node⟩( $2 )→show("‘QUERY_TYPE’:");
    }
#endif
    $$ = $2;
#undef DEBUG_OUTPUT
  }
  ;
```

436. ⟨query variable⟩ ⟶ ⟨variable name⟩ `QUERY TYPE`**.** [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  query_variable: variable_name QUERY_TYPE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `query_variable: variable_name QUERY_TYPE'." << endl;
    if ($2 == 0) {
      temp_strm << "The semantic value of `QUERY_TYPE' == 0.  Not showing." << endl;
    }
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#if 0    /* 1 */
    if ($2 != 0) {
      static_cast<Id_Node>($2)->show("`QUERY_TYPE':");
    }
#endif
#endif
    $$ = $2;
#undef DEBUG_OUTPUT
  }
  ;
```

437. ⟨datasource variable⟩**.** [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
  %type <pointer_value> datasource_variable
```

438. ⟨datasource variable⟩ ⟶ `VARIABLE DATASOURCE TYPE`. [LDF 2006.12.08.]
The semantic value of the `DATASOURCE_TYPE` token is the **Id_Type** object associated with the name specified by the semantic value of the `VARIABLE` token, or 0, if there is no **Id_Type** object. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  datasource_variable: VARIABLE DATASOURCE_TYPE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule ‘datasource_variable: VARIABLE DATASOURCE_TYPE’." ≪ endl;
    if ($2 ≡ 0) {
      temp_strm ≪ "The semantic value of ‘DATASOURCE_TYPE’ == 0.  Not showing." ≪ endl;
    }
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    if ($2 ≠ 0) {
      static_cast⟨Id_Node⟩($2)→show("‘DATASOURCE_TYPE’:");
    }
#endif
    $$ = $2;
#undef DEBUG_OUTPUT
  }
  ;
```

439. ⟨datasource variable⟩ ⟶ ⟨variable name⟩ DATASOURCE TYPE. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datasource_variable: variable_name DATASOURCE_TYPE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `datasource_variable: variable_name DATASOURCE_TYPE'." << endl;
    if ($2 ≡ 0) {
      temp_strm << "The semantic value of `DATASOURCE_TYPE' == 0.  Not showing." << endl;
    }
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
    if ($2 ≠ 0) {
      static_cast<Id_Node>($2)->show("`DATASOURCE_TYPE':");
    }
#endif
    $$ = $2;
#undef DEBUG_OUTPUT
  }
  ;
```

440. ⟨string variable⟩. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> string_variable
```

441. ⟨string variable⟩ ⟶ `VARIABLE STRING TYPE`. [LDF 2006.11.16.]
The semantic value of the `STRING_TYPE` token is the **Id_Type** object associated with the name specified by the semantic value of the `VARIABLE` token, or 0, if there is no **Id_Type** object. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  string_variable: VARIABLE STRING_TYPE
  {
#if 0    /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule `string_variable: VARIABLE STRING_TYPE'." ≪ endl;
    if ($2 ≡ 0) {
      temp_strm ≪ "The semantic value of `STRING_TYPE' == 0.  Not showing." ≪ endl;
    }
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    if ($2 ≠ 0) {
      static_cast⟨Id_Node⟩($2)→show("`STRING_TYPE':");
    }
#endif
    $$ = $2;
#undef DEBUG_OUTPUT
  }
  ;
```

442. ⟨string variable⟩ ⟶ ⟨variable name⟩ STRING TYPE. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  string_variable: variable_name STRING_TYPE
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule `string_variable: variable_name STRING_TYPE'." ≪ endl;
    if ($2 ≡ 0) {
      temp_strm ≪ "The semantic value of `STRING_TYPE' == 0.  Not showing." ≪ endl;
    }
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    if ($2 ≠ 0) {
      static_cast⟨Id_Node⟩($2)→show("`STRING_TYPE':");
    }
#endif
    $$ = $2;
#undef DEBUG_OUTPUT
  }
  ;
```

443. ⟨datetime variable⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this type declaration.

```
⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
  %type <pointer_value> datetime_variable
```

444. ⟨datetime variable⟩ ⟶ `VARIABLE DATETIME TYPE`. [LDF 2006.11.02.]
The semantic value of the `DATETIME_TYPE` token is the **Id_Type** object associated with the name specified by the semantic value of the `VARIABLE` token, or 0, if there is no **Id_Type** object. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datetime_variable: VARIABLE DATETIME_TYPE
{
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'datetime_variable: VARIABLE DATETIME_TYPE'." << endl;
#endif
    $$ = $2;
#undef DEBUG_OUTPUT
}
;
```

445. ⟨datetime variable⟩ ⟶ ⟨variable name⟩ `DATETIME TYPE`. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datetime_variable: variable_name DATETIME_TYPE
  {
#if 1  /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm << endl << "Rule `datetime_variable: variable_name DATETIME_TYPE'." << endl;
    if ($2 ≡ 0) {
      temp_strm << "The semantic value of `DATETIME_TYPE' == 0.  Not showing." << endl;
    }
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm << temp_strm.str();
    temp_strm.str("");
#if 0  /* 1 */
    if ($2 ≠ 0) {
      static_cast⟨Id_Node⟩($2)→show("`DATETIME_TYPE':");
    }
#endif
#endif
    $$ = $2;
#undef DEBUG_OUTPUT
  }
  ;
```

446. Assignment (`assign.w`). [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

Renamed this file. Old name: `passign.w`. New name: `assign.w`.

⟨ `assign.w` 446 ⟩ ≡

```
  static char assign_id_string[] = "$Id: assign.w,v 1.3 2007/01/02 12:36:21 lfinsto1 Exp $";
```

This code is cited in sections 7 and 8.

This code is used in section 642.

447.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type␣&lt;int_value&gt;␣assignment
```

448. ⟨negation optional⟩. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this section.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type␣<int_value>␣negation_optional
```

449. ⟨negation optional⟩ ⟶ EMPTY. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
negation_optional:␣/*␣Empty␣␣*/
{
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘negation_optional:␣/*␣Empty␣␣*/’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = 0;
}
;
```

450. ⟨negation optional⟩ ⟶ `NOT`. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
negation_optional: NOT
  {
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'negation_optional: NOT'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = 1;
  }
  ;
```

451. ⟨match term optional⟩. [LDF 2006.12.11.]

Log

[LDF 2006.12.11.] Added this section.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <int_value> match_term_optional
```

452. ⟨match term optional⟩ ⟶ `EMPTY.` [LDF 2006.12.11.]

Log

[LDF 2006.12.11.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  match_term_optional:␣/*␣Empty␣␣*/
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘match_term_optional:␣/*␣Empty␣␣*/’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = Query_Type::QUERY_TYPE_NULL_TYPE;
  }
  ;
```

453. ⟨match term optional⟩ ⟶ `CONTAINS`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  match_term_optional: CONTAINS
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'match_term_optional: CONTAINS'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = Query_Type::CONTAINS_VALUE;
  }
  ;
```

454. ⟨match term optional⟩ ⟶ `FREETEXT`. [LDF 2006.12.11.]

Log

[LDF 2006.12.11.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  match_term_optional:␣FREETEXT
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘match_term_optional:␣FREETEXT’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = Query_Type::FREETEXT_VALUE;
  }
  ;
```

455. ⟨match term optional⟩ ⟶ `LIKE`. [LDF 2006.12.11.]

Log

[LDF 2006.12.11.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
match_term_optional: LIKE
{
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `match_term_optional: LIKE'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = Query_Type::LIKE_VALUE;
}
;
```

456. ⟨assign or plus-assign⟩. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <int_value> assign_or_plus_assign
```

457. ⟨assign or plus-assign⟩ ⟶ ASSIGN. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  assign_or_plus_assign:␣ASSIGN
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘assign_or_plus_assign:␣ASSIGN’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = ASSIGN;
  }
  ;
```

458. ⟨assign or plus-assign⟩ ⟶ PLUS_ASSIGN. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  assign_or_plus_assign: PLUS_ASSIGN
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `assign_or_plus_assign: PLUS_ASSIGN'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = PLUS_ASSIGN;
  }
  ;
```

459. ⟨assignment operator⟩. [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
  %type <int_value> assignment_operator
```

460. ⟨assignment operator⟩ ⟶ `ASSIGN`. [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  assignment_operator: ASSIGN
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule 'assignment_operator: ASSIGN'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = ASSIGN;
  }
  ;
```

461. ⟨assignment operator⟩ ⟶ PLUS_ASSIGN. [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  assignment_operator:␣PLUS_ASSIGN
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule␣`assignment_operator:␣PLUS_ASSIGN'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = PLUS_ASSIGN;
  }
  ;
```

462. ⟨assignment operator⟩ ⟶ `OR_ASSIGN`. [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  assignment_operator:␣OR_ASSIGN
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘assignment_operator:␣OR_ASSIGN’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = OR_ASSIGN;
  }
  ;
```

463. ⟨assignment operator⟩ ⟶ `XOR_ASSIGN`. [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  assignment_operator:␣XOR_ASSIGN
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘assignment_operator:␣XOR_ASSIGN’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = XOR_ASSIGN;
  }
  ;
```

464. ⟨assignment operator⟩ ⟶ NOT_ASSIGN. [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  assignment_operator:␣NOT_ASSIGN
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘assignment_operator:␣NOT_ASSIGN’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = NOT_ASSIGN;
  }
  ;
```

465. ⟨assignment⟩ ⟶ ⟨query assignment⟩. [LDF 2006.11.02.]

⟨ Parser rules 388 ⟩ +≡

```
  assignment:␣query_assignment
  {
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘assignment:␣query_assignment’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
  }
  ;
```

466. ⟨assignment⟩ ⟶ ⟨datasource assignment⟩. [LDF 2006.12.08.]

Log

[2006.12.08.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
assignment: datasource_assignment
{
  Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
  temp_strm << endl << "Rule 'assignment: datasource_assignment'." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node->log_strm << temp_strm.str();
  temp_strm.str("");
}
;
```

467. ⟨assignment⟩ ⟶ ⟨string assignment⟩. [LDF 2006.11.16.]

Log

[2006.11.16.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
assignment: string_assignment
{
  Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
  temp_strm << endl << "Rule 'assignment: string_assignment'." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node->log_strm << temp_strm.str();
  temp_strm.str("");
}
;
```

468. ⟨query assignment⟩.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> query_assignment
```

469. ⟨query assignment⟩ ⟶ ⟨query variable⟩ `ASSIGN` ⟨query expression⟩. [LDF 2006.11.17.]

Log

[2006.11.17.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  query_assignment:␣query_variable␣ASSIGN␣query_expression
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘query_assignment:␣query_variable␣ASSIGN␣query_expression’." ≪
        endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node id_node = static_cast⟨Id_Node⟩($1);
    if (¬id_node) $$ = 0;
    else {
      if (id_node→value ≠ 0) delete static_cast⟨Query_Node⟩(id_node→value);
      id_node→value = $3;
      $$ = $3;
    }
#undef DEBUG_OUTPUT
  }
  ;
```

470. ⟨assignment⟩ ⟶ ⟨datetime assignment⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  assignment:␣datetime_assignment
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘assignment:␣datetime_assignment’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
  }
  ;
```

471. Targets. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this section.

---

472. ⟨query assignment⟩ ⟶ (etc.). ⟨query assignment⟩ ⟶ ⟨query variable⟩ ⟨assign or plus-assign⟩ `LOCAL DATABASE`. [LDF 2006.11.02.]

Log

[LDF 2006.11.13.] Changed this rule from ⟨query assignment⟩ ⟶ ⟨query variable⟩ `ASSIGN LOCAL` to ⟨query assignment⟩ ⟶ ⟨query variable⟩ `ASSIGN LOCAL DATABASE`.

[LDF 2006.11.13.] Changed `ASSIGN` to ⟨assign or plus-assign⟩.

```
⟨ Parser rules 388 ⟩ +≡
  query_assignment:␣query_variable␣assign_or_plus_assign␣LOCAL␣DATABASE
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘query_assignment:␣query_variable␣assign_or_plus_assig\
        n␣LOCAL␣DATABASE’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = query_assignment_func_0(parameter, $1, $2, LOCAL, DATABASE, 0, 0);
#undef DEBUG_OUTPUT
  }
  ;
```

473. ⟨query assignment⟩ ⟶ (etc.). ⟨query assignment⟩ ⟶ ⟨query variable⟩ ⟨assign or plus-assign⟩ `LOCAL` `SERVER`. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this rule.

[LDF 2006.11.13.] Changed `ASSIGN` to ⟨assign or plus-assign⟩.

```
⟨ Parser rules 388 ⟩ +≡
  query_assignment:␣query_variable␣assign_or_plus_assign␣LOCAL␣SERVER
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘query_assignment:␣query_variable␣assign_or_plus_assig\
        n␣LOCAL␣SERVER’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = query_assignment_func_0(parameter, $1, $2, LOCAL, SERVER, 0, 0);
#undef DEBUG_OUTPUT
  }
  ;
```

474. ⟨query assignment⟩ ⟶ **(etc.).** ⟨query assignment⟩ ⟶ ⟨query variable⟩ ⟨assign or plus-assign⟩ `REMOTE DATABASE`. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this rule.

[LDF 2006.11.13.] Changed `ASSIGN` to ⟨assign or plus-assign⟩.

```
⟨ Parser rules 388 ⟩ +≡
  query_assignment: query_variable assign_or_plus_assign REMOTE DATABASE
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'query_assignment: query_variable assign_or_plus_assig\
        n REMOTE DATABASE'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = query_assignment_func_0(parameter, $1, ASSIGN, REMOTE, DATABASE, 0, 0);
#undef DEBUG_OUTPUT
  }
  ;
```

475. ⟨query assignment⟩ ⟶ (etc.). ⟨query assignment⟩ ⟶ ⟨query variable⟩ ⟨assign or plus-assign⟩ `REMOTE SERVER`. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this rule.

[LDF 2006.11.13.] Changed `ASSIGN` to ⟨assign or plus-assign⟩.

⟨ Parser rules 388 ⟩ +≡

```
  query_assignment:␣query_variable␣assign_or_plus_assign␣REMOTE␣SERVER
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘query_assignment:␣query_variable␣assign_or_plus_assig\
        n␣REMOTE␣SERVER’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = query_assignment_func_0(parameter, $1, $2, REMOTE, SERVER, 0, 0);
#undef DEBUG_OUTPUT
  }
  ;
```

476. Fields. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this section.

---

477. ⟨query assignment⟩ ⟶ **(etc.).** ⟨query assignment⟩ ⟶ ⟨query variable⟩ `COLON` ⟨field specifier⟩ ⟨assignment operator⟩ ⟨negation optional⟩ ⟨match term optional⟩ ⟨query expression⟩. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

[LDF 2006.11.21.] Now setting *curr_id_node→value→id_node* and *curr_id_node→value→name*.

[LDF 2006.12.19.] Removed code from this rule. Now calling **Scan_Parse** :: *query_assignment_func_1* .

```
⟨ Parser rules 388 ⟩ +≡
  query_assignment:␣query_variable␣COLON␣field_specifier␣ assignment_operator␣negation_optional␣match_term_optional␣query_expression {
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
# # undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "Rule␣‘query_assignment:␣query_variable␣COLON␣field_specifier␣" ≪
        "assignment_operator␣negation_optional␣match_term_optional␣" ≪
        "query_expression’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    Id_Node curr_id_node = static_cast⟨Id_Node⟩( $1 );
    int status = query_assignment_func_1 (scanner_node, curr_id_node, $3 , $4 , $5 , $6 , $7 ,
        QUERY_TYPE);
    $$ = static_cast⟨void *⟩(curr_id_node);
```

478. If **Scan_Parse** :: *query_assignment_func_1* fails, *curr_id_node* will be 0 anyway, so it's not really necessary to check the value of *status*. It's only done for the sake of debugging output, which is conditionally compiled. [LDF 2006.12.19.]

```
⟨ Parser rules 388 ⟩ +≡
#ifdef DEBUG_OUTPUT
  if (status ≠ 0) {
    temp_strm ≪ "ERROR!␣␣In␣‘yyparse’," ≪ endl ≪
        "rule␣‘query_assignment:␣query_variable␣COLON␣field_specifier␣" ≪
        endl ≪ "assignment_operator␣negation_optional␣match_term_optional␣" ≪
        "query_expression’:" ≪ endl ≪ "‘Scan_Parse::query_assignment_func_1’␣f\
        ailed,␣returning␣" ≪ status ≪ "." ≪ endl ≪ "Setting␣‘query_assignment’␣to␣0." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    $$ = 0;
  }    /* status ≠ 0 */
#endif
```

479.

```
⟨ Parser rules 388 ⟩ +≡
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "Exiting␣rule␣‘query_assignment:␣query_variable␣COLON␣field_\
      specifier␣" ≪ "assignment_operator␣negation_optional␣match_term_optional␣" ≪
      "query_expression’." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
#undef DEBUG_OUTPUT
  } ;
```

480. ⟨query assignment⟩ ⟶ **(etc.).** ⟨query assignment⟩ ⟶ ⟨query variable⟩ `COLON` ⟨field specifier⟩ ⟨assignment operator⟩ ⟨negation optional⟩ ⟨match term optional⟩ ⟨query expression⟩. [LDF 2006.12.19.]

Log

[LDF 2006.12.19.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
query_assignment:␣query_variable␣COLON␣field_specifier␣ assignment_operator␣negation_optional␣match_term_optional␣datetime_expression {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
# # undef DEBUG_OUTPUT
#endif
   Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
#ifdef DEBUG_OUTPUT
   temp_strm ≪ endl ≪ "Rule␣`query_assignment:␣query_variable␣COLON␣field_specifier␣" ≪
       "assignment_operator␣negation_optional␣match_term_optional␣" ≪
       "datetime_expression'." ≪ endl;
   cerr_mutex.Lock();
   cerr ≪ temp_strm.str();
   cerr_mutex.Unlock();
   scanner_node→log_strm ≪ temp_strm.str();
   temp_strm.str("");
#endif
   Id_Node curr_id_node = static_cast⟨Id_Node⟩($1);
   int status = query_assignment_func_1(scanner_node, curr_id_node, $3, $4, $5, $6, $7,
       DATETIME_TYPE);
   $$ = static_cast⟨void *⟩(curr_id_node);
```

481. If **Scan_Parse** :: *query_assignment_func_1* fails, *curr_id_node* will be 0 anyway, so it's not really necessary to check the value of *status*. It's only done for the sake of debugging output, which is conditionally compiled. [LDF 2006.12.19.]

⟨ Parser rules 388 ⟩ +≡

```
#ifdef DEBUG_OUTPUT
  if (status ≠ 0) {
    temp_strm ≪ "ERROR!␣␣In␣`yyparse'," ≪ endl ≪
       "rule␣`query_assignment:␣query_variable␣COLON␣field_specifier␣" ≪
       endl ≪ "assignment_operator␣negation_optional␣match_term_optional␣" ≪
       "datetime_expression':" ≪ endl ≪ "`Scan_Parse::query_assignment_func_1'␣f\
       ailed,␣returning␣" ≪ status ≪ "." ≪ endl ≪ "Setting␣`query_assignment'␣to␣0." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    $$ = 0;
  }   /* status ≠ 0 */
#endif
```

482.

⟨ Parser rules 388 ⟩ +≡
```
#ifdef DEBUG_OUTPUT
  temp_strm << endl << "Exiting rule `query_assignment: query_variable COLON field_\
      specifier " << "assignment_operator negation_optional match_term_optional " <<
      "datetime_expression'." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node->log_strm << temp_strm.str();
  temp_strm.str("");
#endif
#undef DEBUG_OUTPUT
  };
```

483. ⟨field specifier⟩. [LDF 2006.12.13.]

Log

[LDF 2006.12.13.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
```
  %type <pointer_value> field_specifier
```

484. ⟨field specifier⟩ ⟶ ⟨field designator⟩ ⟨field qualifier list⟩. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡
```
  field_specifier: field_designator field_qualifier_list {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
      Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
      temp_strm << endl << "Rule `field_specifier: field_designator field_qualifier_list'." <<
          endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node->log_strm << temp_strm.str();
      temp_strm.str("");
#endif
      deque < int > *d = ($2 == 0) ? new deque < int > : static_cast < deque < int >*> ($2);
      d->push_front($1);
      $$ = static_cast<void *>(d); } ;
```

485. ⟨field designator⟩. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type␣&lt;int_value&gt;␣field_designator
```

486. ⟨field designator⟩ ⟶ ACCESS_NUMBER. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
field_designator:␣ACCESS_NUMBER
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule␣`field_designator:␣ACCESS_NUMBER'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = ACCESS_NUMBER;
  }
  ;
```

487. ⟨field designator⟩ ⟶ `AUTHOR`. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_designator: AUTHOR
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `field_designator: AUTHOR'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = AUTHOR;
  }
  ;
```

488. ⟨field designator⟩ ⟶ BIBLIOGRAPHIC_TYPE. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_designator: BIBLIOGRAPHIC_TYPE
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `field_designator: BIBLIOGRAPHIC_TYPE'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = BIBLIOGRAPHIC_TYPE;
  }
  ;
```

489. ⟨field designator⟩ ⟶ `CALL_NUMBER`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣CALL_NUMBER
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣CALL_NUMBER’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = CALL_NUMBER;
  }
  ;
```

490. ⟨field designator⟩ ⟶ CLASSIFICATION. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator: CLASSIFICATION
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm << endl << "Rule 'field_designator: CLASSIFICATION'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = CLASSIFICATION;
  }
  ;
```

491. ⟨field designator⟩ ⟶ `COMPANY`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣COMPANY
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣COMPANY’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = COMPANY;
  }
  ;
```

492. ⟨field designator⟩ ⟶ `CONTENT_SUMMARY`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣CONTENT_SUMMARY
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣CONTENT_SUMMARY’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = CONTENT_SUMMARY;
  }
  ;
```

493. ⟨field designator⟩ ⟶ `CONTRIBUTOR`. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣CONTRIBUTOR
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣CONTRIBUTOR’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = CONTRIBUTOR;
  }
  ;
```

494. ⟨field designator⟩ ⟶ `CREATOR`. [LDF 2006.12.05.]

Log

[LDF 2006.12.05.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣CREATOR
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣CREATOR’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = CREATOR;
  }
  ;
```

495. ⟨field designator⟩ ⟶ `DATABASE_PROVIDER`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣DATABASE_PROVIDER
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣DATABASE_PROVIDER’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = DATABASE_PROVIDER;
  }
  ;
```

496. ⟨field designator⟩ ⟶ DESCRIPTION. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_designator: DESCRIPTION
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `field_designator: DESCRIPTION'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = DESCRIPTION;
  }
  ;
```

497. ⟨field designator⟩ ⟶ `EXEMPLAR_PRODUCTION_NUMBER`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_designator: EXEMPLAR_PRODUCTION_NUMBER
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'field_designator: EXEMPLAR_PRODUCTION_NUMBER'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = EXEMPLAR_PRODUCTION_NUMBER;
  }
  ;
```

498. ⟨field designator⟩ ⟶ IDENTIFIER. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_designator: IDENTIFIER
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'field_designator: IDENTIFIER'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = IDENTIFIER;
  }
  ;
```

499. ⟨field designator⟩ ⟶ `INSTITUTION`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_designator:␣INSTITUTION
  {
#if 1      /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣INSTITUTION’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = INSTITUTION;
  }
  ;
```

500. ⟨field designator⟩ ⟶ LANGUAGE. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣LANGUAGE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣LANGUAGE’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = LANGUAGE;
  }
  ;
```

501. ⟨field designator⟩ ⟶ `MAIN_CANONICAL_TITLE`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣MAIN_CANONICAL_TITLE
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣MAIN_CANONICAL_TITLE’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = MAIN_CANONICAL_TITLE;
  }
  ;
```

502. ⟨field designator⟩ ⟶ PERMUTATION_PATTERN. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣PERMUTATION_PATTERN
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣`field_designator:␣PERMUTATION_PATTERN'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = PERMUTATION_PATTERN;
  }
  ;
```

503. ⟨field designator⟩ ⟶ `PERSON`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
   field_designator:␣PERSON
   {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
      Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
      temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣PERSON’." ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
      $$ = PERSON;
   }
   ;
```

504. ⟨field designator⟩ ⟶ PHYSICAL_DESCRIPTION. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣PHYSICAL_DESCRIPTION
  {
#if 1  /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣PHYSICAL_DESCRIPTION’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = PHYSICAL_DESCRIPTION;
  }
  ;
```

505. ⟨field designator⟩ ⟶ PUBLISHER. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣PUBLISHER
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣PUBLISHER’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = PUBLISHER;
  }
  ;
```

506. ⟨field designator⟩ ⟶ `RECORD`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
   field_designator:␣RECORD
   {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
     Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
     temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣RECORD’." ≪ endl;
     cerr_mutex.Lock();
     cerr ≪ temp_strm.str();
     cerr_mutex.Unlock();
     scanner_node→log_strm ≪ temp_strm.str();
     temp_strm.str("");
#endif
     $$ = RECORD;
   }
   ;
```

507. ⟨field designator⟩ ⟶ `REMOTE_ACCESS`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator: REMOTE_ACCESS
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣REMOTE_ACCESS’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = REMOTE_ACCESS;
  }
  ;
```

508. ⟨field designator⟩ ⟶ `RIGHTS`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator: RIGHTS
  {
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule 'field_designator: RIGHTS'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = RIGHTS;
  }
  ;
```

509. ⟨field designator⟩ ⟶ `SOURCE`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_designator: SOURCE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `field_designator: SOURCE'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = SOURCE;
  }
  ;
```

510. ⟨field designator⟩ ⟶ `SUBJECT`. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣SUBJECT
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣SUBJECT’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = SUBJECT;
  }
  ;
```

511. ⟨field designator⟩ ⟶ `SUPERORDINATE_ENTITIES`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣SUPERORDINATE_ENTITIES
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣SUPERORDINATE_ENTITIES’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = SUPERORDINATE_ENTITIES;
  }
  ;
```

512. ⟨field designator⟩ ⟶ `TITLE`. [LDF 2006.11.14.]

Log

[2006.11.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_designator:␣TITLE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣`field_designator:␣TITLE'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = TITLE;
  }
  ;
```

513. ⟨field designator⟩ ⟶ `TYPE`. [LDF 2006.12.12.]

Log

[LDF 2006.12.12.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
field_designator:␣TYPE
{
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_designator:␣TYPE’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = TYPE;
}
;
```

514. ⟨field qualifier⟩. [LDF 2006.12.13.]

Log

[LDF 2006.12.13.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type␣<pointer_value>␣field_qualifier_list
```

515. ⟨field qualifier list⟩ ⟶ `EMPTY`. [LDF 2006.12.13.]

Log

[LDF 2006.12.13.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier_list: /* Empty  */ {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule 'field_qualifier: EMPTY'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    deque < int > *d = new deque < int > ;
    $$ = static_cast⟨void *⟩(d); } ;
```

516. ⟨field qualifier list⟩ ⟶ ⟨field qualifier list⟩ `PERIOD` ⟨field qualifier⟩. [LDF 2006.12.13.]

Log

[LDF 2006.12.13.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier_list: field_qualifier_list PERIOD field_qualifier {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule 'field_qualifier: field_qualifier_list PERIOD field_qu\
        alifier'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    deque < int > *d = static_cast ⟨ deque < int >*> ( $1 );
    d→push_back( $3 );
    $$ = static_cast⟨void *⟩(d); } ;
```

517. ⟨field qualifier⟩**.** [LDF 2006.12.07.]

Log

[LDF 2006.12.07.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <int_value> field_qualifier
```

518. Generic. [LDF 2006.12.13.]

Log

[2006.12.13.] Added this section.

---

519. ⟨field qualifier⟩ ⟶ ID**.** [LDF 2006.12.13.]

Log

[LDF 2006.12.13.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
field_qualifier: ID
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `field_qualifier: ID'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = ID;
  }
  ;
```

520. Names. [LDF 2006.12.13.]

Log

[2006.12.13.] Added this section.

---

521. ⟨field qualifier⟩ ⟶ `SURNAME`. [LDF 2006.12.07.]

Log

[LDF 2006.12.07.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣SURNAME
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣SURNAME’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = SURNAME;
  }
  ;
```

522. ⟨field qualifier⟩ ⟶ `GIVEN_NAME`. [LDF 2006.12.13.]

Log

[LDF 2006.12.13.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣GIVEN_NAME
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣`field_qualifier:␣GIVEN_NAME'." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = GIVEN_NAME;
  }
  ;
```

523. ⟨field qualifier⟩ ⟶ `PREFIX`. [LDF 2006.12.13.]

Log

[LDF 2006.12.13.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣PREFIX
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣PREFIX’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = PREFIX;
  }
  ;
```

524. Field qualifiers for database table columns. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

---

525. Records. [LDF 2006.12.14.]

Log

[2006.12.14.] Added this section.

---

526. ⟨field qualifier⟩ ⟶ `ELN_ORIGINAL_ENTRY`. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣ELN_ORIGINAL_ENTRY
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣ELN_ORIGINAL_ENTRY’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = ELN_ORIGINAL_ENTRY;
  }
  ;
```

527. ⟨field qualifier⟩ ⟶ ELN_MOST_RECENT_CHANGE. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_qualifier:␣ELN_MOST_RECENT_CHANGE
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣ELN_MOST_RECENT_CHANGE’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = ELN_MOST_RECENT_CHANGE;
  }
  ;
```

528. ⟨field qualifier⟩ ⟶ `ELN_STATUS_CHANGE`. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_qualifier:␣ELN_STATUS_CHANGE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣ELN_STATUS_CHANGE’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = ELN_STATUS_CHANGE;
  }
  ;
```

529. ⟨field qualifier⟩ ⟶ IDENTIFICATION_NUMBER. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣IDENTIFICATION_NUMBER
  {
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣IDENTIFICATION_NUMBER’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = IDENTIFICATION_NUMBER;
  }
  ;
```

530. ⟨field qualifier⟩ ⟶ `DATE_ORIGINAL_ENTRY`. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣DATE_ORIGINAL_ENTRY
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣DATE_ORIGINAL_ENTRY’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = DATE_ORIGINAL_ENTRY;
  }
  ;
```

531. ⟨field qualifier⟩ ⟶ DATE_MOST_RECENT_CHANGE. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  field_qualifier:␣DATE_MOST_RECENT_CHANGE
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣DATE_MOST_RECENT_CHANGE’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = DATE_MOST_RECENT_CHANGE;
  }
  ;
```

532. ⟨field qualifier⟩ ⟶ `DATE_STATUS_CHANGE`. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier: DATE_STATUS_CHANGE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule 'field_qualifier: DATE_STATUS_CHANGE'." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = DATE_STATUS_CHANGE;
  }
  ;
```

533. ⟨field qualifier⟩ ⟶ `SOURCE_ID.` [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣SOURCE_ID
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣SOURCE_ID’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = SOURCE_ID;
  }
  ;
```

534. ⟨field qualifier⟩ ⟶ `YEAR_APPEARANCE_BEGIN`. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣YEAR_APPEARANCE_BEGIN
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣YEAR_APPEARANCE_BEGIN’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = YEAR_APPEARANCE_BEGIN;
  }
  ;
```

535. ⟨field qualifier⟩ ⟶ YEAR_APPEARANCE_END. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣YEAR_APPEARANCE_END
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣YEAR_APPEARANCE_END’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = YEAR_APPEARANCE_END;
  }
  ;
```

536. ⟨field qualifier⟩ ⟶ `YEAR_APPEARANCE_RAK_WB`. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  field_qualifier:␣YEAR_APPEARANCE_RAK_WB
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘field_qualifier:␣YEAR_APPEARANCE_RAK_WB’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = YEAR_APPEARANCE_RAK_WB;
  }
  ;
```

537. ⟨field qualifier⟩ ⟶ `YEAR_APPEARANCE_ORIGINAL`. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
field_qualifier:␣YEAR_APPEARANCE_ORIGINAL
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule␣‘field_qualifier:␣YEAR_APPEARANCE_ORIGINAL’." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = YEAR_APPEARANCE_ORIGINAL;
  }
  ;
```

538. ⟨datasource assignment⟩. [LDF 2006.12.08.]

Log

[2006.12.08.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type␣<pointer_value>␣datasource_assignment
```

539. ⟨datasource assignment⟩ ⟶ ⟨datasource variable⟩ `ASSIGN` ⟨datasource expression⟩. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datasource_assignment: datasource_variable ASSIGN datasource_expression
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule ‘datasource_assignment: datasource_variable " ≪
        "ASSIGN datasource_expression’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node id_node = static_cast⟨Id_Node⟩($1);
    if (¬id_node) $$ = 0;
    else {
      if (id_node→value ≠ 0) delete static_cast⟨Datasource_Node⟩(id_node→value);
      id_node→value = $3;
      $$ = $3;
    }
#undef DEBUG_OUTPUT
  }
  ;
```

540. ⟨datasource assignment⟩ ⟶ ⟨datasource variable⟩ `ASSIGN DBT`. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datasource_assignment:␣datasource_variable␣ASSIGN␣DBT
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datasource_assignment:␣datasource_variable␣ASSIGN␣DBT’." ≪
        endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node id_node = static_cast⟨Id_Node⟩( $1 );
    if (¬id_node) $$ = 0;
    else {
      if (id_node→value ≡ 0) id_node→value = static_cast⟨void *⟩(new Datasource_Type);
      static_cast⟨Datasource_Node⟩(id_node→value)→type = Datasource_Type::DBT_TYPE;
      $$ = id_node→value;
    }
#undef DEBUG_OUTPUT
  }
  ;
```

541. ⟨datasource assignment⟩ ⟶ ⟨datasource variable⟩ `ASSIGN GBV_GVK`. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datasource_assignment: datasource_variable ASSIGN GBV_GVK
{
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'datasource_assignment: datasource_variable ASSIGN GBV\
        _GVK'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node id_node = static_cast<Id_Node>($1);
    if (!id_node) $$ = 0;
    else {
      if (id_node->value == 0) id_node->value = static_cast<void *>(new Datasource_Type);
      static_cast<Datasource_Node>(id_node->value)->type = Datasource_Type::GBV_GVK_TYPE;
      $$ = id_node->value;
    }
#undef DEBUG_OUTPUT
}
;
```

542. ⟨datasource assignment⟩ ⟶ ⟨datasource variable⟩ `ASSIGN TIMMS`. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datasource_assignment: datasource_variable ASSIGN TIMMS
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datasource_assignment:␣datasource_variable␣ASSIGN␣TIMMS’." ≪
        endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node id_node = static_cast⟨Id_Node⟩($1);
    if (¬id_node) $$ = 0;
    else {
      if (id_node→value ≡ 0) id_node→value = static_cast⟨void *⟩(new Datasource_Type);
      static_cast⟨Datasource_Node⟩(id_node→value)→type = Datasource_Type::TIMMS_TYPE;
      $$ = id_node→value;
    }
#undef DEBUG_OUTPUT
  }
  ;
```

543. ⟨datasource assignment⟩ ⟶ ⟨datasource variable⟩ `ASSIGN DATASOURCE_FILE`. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datasource_assignment: datasource_variable ASSIGN DATASOURCE_FILE
  {
#if 1      /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule `datasource_assignment: datasource_variable ASSIGN DAT\
        ASOURCE_FILE'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node id_node = static_cast⟨Id_Node⟩( $1 );
    if (¬id_node) $$ = 0;
    else {
      if (id_node→value ≡ 0) id_node→value = static_cast⟨void *⟩(new Datasource_Type);
      static_cast⟨Datasource_Node⟩(id_node→value)→type =
          Datasource_Type::DATASOURCE_FILE_TYPE;
      $$ = id_node→value;
    }
#undef DEBUG_OUTPUT
  }
  ;
```

544. ⟨string assignment⟩. [LDF 2006.11.16.]

Log

[2006.11.16.] Added this type declaration.

```
⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
  %type <string_value> string_assignment
```

545. ⟨string assignment⟩ ⟶ ⟨string variable⟩ `ASSIGN` ⟨string expression⟩. [LDF 2006.11.16.]

Log

[2006.11.16.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  string_assignment:␣string_variable␣ASSIGN␣string_expression
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "Rule␣‘string_assignment:␣␣string_variable␣" ≪
        "ASSIGN␣string_expression’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node curr_id_node = static_cast⟨Id_Node⟩( $1 );
    if (curr_id_node ≡ 0) {
      temp_strm ≪ "WARNING!␣␣In␣‘yyparse’,␣rule␣" ≪
          "‘string_assignment:␣␣string_variable" ≪ endl ≪ "ASSIGN␣string_expression’:" ≪
          endl ≪ "‘string_variable’␣==␣0.␣␣Can’t␣assign." ≪ endl ≪
          "Type␣<RETURN>␣to␣continue:␣";
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      getchar();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
      strcpy( $$ , $3 );
    }
    else {
      if (curr_id_node→value ≡ 0) curr_id_node→value = new string;
      *static_cast⟨string *⟩(curr_id_node→value) = $3 ;
      strcpy( $$ , $3 );
    }
#undef DEBUG_OUTPUT
  }
  ;
```

546. ⟨string assignment⟩ ⟶ ⟨string variable⟩ `ASSIGN TEX` ⟨query expression⟩. [LDF 2006.11.16.]

Log

[2006.11.16.] Added this rule.

[LDF 2006.12.01.] !! BUG FIX: Added code for the case that *curr_id_node→value* ≠ 0.

⟨ Parser rules 388 ⟩ +≡

```
string_assignment: string_variable ASSIGN TEX query_expression {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Rule `string_assignment: string_variable ASSIGN TEX query_e\
        xpression" << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node curr_id_node = static_cast<Id_Node>($1);
    Query_Node q = static_cast<Query_Node>($4);
    string *curr_string = 0;
    stringstream tex_strm;
    if (curr_id_node == 0 || q == 0) {
      strcpy($$, "");
    }
    else    /* curr_id_node != 0 */
    { curr_id_node->subtype = Id_Type::TEX_STRING_TYPE;
    if (curr_id_node->value == 0) {
      curr_string = new string;
      curr_id_node->value = static_cast<void *>(curr_string);
    }
    else {
      *static_cast<string *>(curr_id_node->value) = "";
      curr_string = static_cast<string *>(curr_id_node->value);
    }
```

547.

⟨ Parser rules 388 ⟩ +≡

```
  stringstream *tex_strm = new stringstream;
  *curr_string = q->generate_tex_string(scanner_node, tex_strm);
  tex_strm->str("");
  strcpy($$, curr_string->c_str()); }    /* else (curr_id_node != 0) */
  } ;
```

548. ⟨string assignment⟩ ⟶ ⟨string variable⟩ ASSIGN SQL OAI ⟨query expression⟩. [LDF 2006.12.01.]

Log

[LDF 2006.12.01.] Added this rule.

[LDF 2006.12.06.] Changed this rule from ⟨string assignment⟩ ⟶ ⟨string variable⟩ ASSIGN SQL ⟨query expression⟩ to ⟨string assignment⟩ ⟶ ⟨string variable⟩ ASSIGN SQL OAI ⟨query expression⟩.

⟨ Parser rules 388 ⟩ +≡

```
  string_assignment: string_variable ASSIGN SQL OAI query_expression {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Rule `string_assignment: string_variable ASSIGN " <<
        "SQL OAI query_expression" << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    stringstream tex_strm;
    stringstream select_strm;
    stringstream from_strm;
    stringstream where_strm_0;
    stringstream where_strm_1;
    Id_Node curr_id_node = static_cast<Id_Node>($1);
    Query_Node q = static_cast<Query_Node>($5);
    string *curr_string = 0;
    if (curr_id_node == 0 || q == 0) {
      strcpy($$, "");
    }
    else   /* curr_id_node != 0 */
    { curr_id_node->subtype = Id_Type::SQL_STRING_TYPE;
    if (curr_id_node->value == 0) {
      curr_string = new string;
      curr_id_node->value = static_cast<void *>(curr_string);
    }
    else {
      *static_cast<string *>(curr_id_node->value) = "";
      curr_string = static_cast<string *>(curr_id_node->value);
    }
```

549.

⟨ Parser rules 388 ⟩ +≡
```
bool first_select = true;
bool first_from = true;
bool first_where = true;
bitset < QUERY_TYPE_BITSET_SIZE > field_flags(0L);
int r = q→generate_sql_string(scanner_node, OAI, &tex_strm, &select_strm, &from_strm, &where_strm_0,
      &where_strm_1, &first_select, &first_from, &first_where, &field_flags, curr_string);
tex_strm.str("");
if (r ≡ 0) strcpy($$, curr_string→c_str());
else strcpy($$, "");
}   /* else (curr_id_node ≠ 0) */
};
```

550. ⟨string assignment⟩ ⟶ ⟨string variable⟩ `ASSIGN SQL PICA` ⟨query expression⟩. [LDF 2006.12.01.]

Log

[LDF 2006.12.06.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  string_assignment: string_variable ASSIGN SQL PICA query_expression {
#if 1  /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Rule `string_assignment: string_variable ASSIGN " <<
        "SQL PICA query_expression" << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    stringstream tex_strm;
    stringstream select_strm;
    stringstream from_strm;
    stringstream where_strm_0;
    stringstream where_strm_1;
    Id_Node curr_id_node = static_cast<Id_Node>($1);
    Query_Node q = static_cast<Query_Node>($5);
    string *curr_string = 0;
    if (curr_id_node == 0 || q == 0) {
      strcpy($$, "");
    }
    else  /* curr_id_node != 0 */
    { curr_id_node->subtype = Id_Type::SQL_STRING_TYPE;
    if (curr_id_node->value == 0) {
      curr_string = new string;
      curr_id_node->value = static_cast<void *>(curr_string);
    }
    else {
      *static_cast<string *>(curr_id_node->value) = "";
      curr_string = static_cast<string *>(curr_id_node->value);
    }
```

551.

⟨ Parser rules 388 ⟩ +≡

```
bool first_select = true;
bool first_from = true;
bool first_where = true;
bitset < QUERY_TYPE_BITSET_SIZE > field_flags(0L);
int r = q->generate_sql_string(scanner_node, PICA, &tex_strm, &select_strm, &from_strm, &where_strm_0,
      &where_strm_1, &first_select, &first_from, &first_where, &field_flags, curr_string);
tex_strm.str("");
if (r ≡ 0) strcpy($$, curr_string->c_str());
else strcpy($$, "");
}    /* else (curr_id_node ≠ 0) */
};
```

552. ⟨datetime assignment⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> datetime_assignment
```

553. ⟨datetime assignment⟩ ⟶ ⟨datetime variable⟩ `ASSIGN` ⟨datetime expression⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datetime_assignment: datetime_variable ASSIGN datetime_expression
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'datetime_assignment: datetime_variable ASSIGN datetim\
        e_expression'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node id_node = static_cast<Id_Node>($1);
    if (!id_node) $$ = 0;
    else {
      if (id_node->value != 0) delete static_cast<Date_Time_Node>(id_node->value);
      id_node->value = $3;
      $$ = $3;
    }
#undef DEBUG_OUTPUT
  }
  ;
```

554. ⟨datetime assignment⟩ ⟶ ⟨datetime variable⟩ COLON ⟨datetime specifier⟩ **(etc.).** ⟨datetime assignment⟩ ⟶ ⟨datetime variable⟩ COLON ⟨datetime specifier⟩ ⟨assignment operator⟩ INTEGER. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  datetime_assignment: datetime_variable COLON datetime_specifier
  assignment_operator INTEGER
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'datetime_assignment: datetime_variable COLON " <<
        "datetime_specifier assignment_operator INTEGER'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node id_node = static_cast<Id_Node>($1);
    if (!id_node) {
      temp_strm << "ERROR!  In 'yyparse', rule" << endl <<
          "'datetime_assignment: datetime_variable COLON " << "datetime_specifier \
          assignment_operator INTEGER':" << endl << "'datetime_variable' == NULL.  " <<
          "Setting value of action ('datetime_assignment') " << "to 0." << endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node->log_strm << temp_strm.str();
      temp_strm.str("");
      $$ = 0;
    }    /* if (!id_node) */
    else    /* id_node != 0 */
    {
      Date_Time_Node d = static_cast<Date_Time_Node>(id_node->value);
      int val = $5;
      datetime_assignment_func_0(scanner_node, d, $3, $4, static_cast<void *>(&val), INTEGER);
      id_node->value = static_cast<void *>(d);
      $$ = id_node->value;
    }    /* else (id_node != 0) */
#undef DEBUG_OUTPUT
  }
  ;
```

555. ⟨datetime assignment⟩ ⟶ ⟨datetime variable⟩ COLON ⟨datetime specifier⟩ **(etc.).** ⟨datetime assignment⟩ ⟶ ⟨datetime variable⟩ COLON ⟨datetime specifier⟩ ⟨assignment operator⟩ FLOAT. [LDF 2006.12.18.]

Log

[2006.12.18.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_assignment:␣datetime_variable␣COLON␣datetime_specifier␣
  assignment_operator␣FLOAT
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_assignment:␣datetime_variable␣COLON␣" ≪
        "datetime_specifier␣assignment_operator␣FLOAT’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    Id_Node id_node = static_cast⟨Id_Node⟩( $1 );
    if (¬id_node) {
      temp_strm ≪ "ERROR!␣␣In␣‘yyparse’,␣rule" ≪ endl ≪
          "‘datetime_assignment:␣datetime_variable␣COLON␣" ≪ "datetime_specifier␣\
          assignment_operator␣FLOAT’:" ≪ endl ≪ "‘datetime_variable’␣==␣NULL.␣␣" ≪
          "Setting␣value␣of␣action␣(‘datetime_assignment’)␣" ≪ "to␣0." ≪ endl;
      cerr_mutex.Lock( );
      cerr ≪ temp_strm.str( );
      cerr_mutex.Unlock( );
      scanner_node→log_strm ≪ temp_strm.str( );
      temp_strm.str("");
      $$ = 0;
    }   /* if (¬id_node) */
    else   /* id_node ≠ 0 */
    {
      Date_Time_Node d = static_cast⟨Date_Time_Node⟩(id_node→value);
      float val = $5;
      datetime_assignment_func_0(scanner_node, d, $3, $4, static_cast⟨void *⟩(&val), FLOAT);
      id_node→value = static_cast⟨void *⟩(d);
      $$ = id_node→value;
    }   /* else (id_node ≠ 0) */
#undef DEBUG_OUTPUT
  }
  ;
```

556. Datetime specifier.

Log

[2006.12.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type␣&lt;int_value&gt;␣datetime_specifier
```

557. ⟨datetime specifier⟩ ⟶ `YEAR_RANGE_BEGIN`. [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datetime_specifier:␣YEAR_RANGE_BEGIN
{
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_specifier:␣YEAR_RANGE_BEGIN’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = YEAR_RANGE_BEGIN;
}
;
```

558. ⟨datetime specifier⟩ ⟶ YEAR_RANGE_END. [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_specifier:␣YEAR_RANGE_END
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_specifier:␣YEAR_RANGE_END’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = YEAR_RANGE_END;
  }
  ;
```

559. ⟨datetime specifier⟩ ⟶ `YEAR`. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_specifier:␣YEAR
  {
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_specifier:␣YEAR’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = YEAR;
  }
  ;
```

560. ⟨datetime specifier⟩ ⟶ MONTH. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  datetime_specifier: MONTH
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `datetime_specifier: MONTH'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = MONTH;
  }
  ;
```

561. ⟨datetime specifier⟩ ⟶ `DAY`. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_specifier:␣DAY
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣`datetime_specifier:␣DAY'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = DAY;
  }
  ;
```

562. ⟨datetime specifier⟩ ⟶ HOUR. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_specifier:␣HOUR
  {
#if 1  /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_specifier:␣HOUR’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = HOUR;
  }
  ;
```

563. ⟨datetime specifier⟩ ⟶ `MINUTE`. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_specifier:␣MINUTE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm << endl << "Rule␣`datetime_specifier:␣MINUTE'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = MINUTE;
  }
  ;
```

564. ⟨datetime specifier⟩ ⟶ `SECOND`. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_specifier:␣SECOND
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_specifier:␣SECOND’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = SECOND;
  }
  ;
```

565. Commands (`commands.w`). [LDF 2006.11.13.]

Log

[LDF 2006.11.13.] Added this file.

```
⟨ commands.w 565 ⟩ ≡
  static char commands_id_string[] = "$Id:␣commands.w,v␣1.3␣2007/01/02␣12:36:\
      41␣lfinsto1␣Exp␣$";
```

This code is cited in sections 7 and 8.

This code is used in section 642.

566.

```
⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
  %type␣<int_value>␣command
```

567. Messages. [LDF 2006.11.15.]

Log

[2006.11.15.] Added this section.

568. ⟨message or errmessage⟩. [LDF 2006.11.15.]

Log

[2006.11.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <int_value> message_or_errmessage
```

569. ⟨message or errmessage⟩ ⟶ MESSAGE. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
message_or_errmessage: MESSAGE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `message_or_errmessage: MESSAGE'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = MESSAGE;
  }
  ;
```

570. ⟨message or errmessage⟩ ⟶ `ERRMESSAGE`. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  message_or_errmessage: ERRMESSAGE
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule ‘message_or_errmessage: ERRMESSAGE’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = ERRMESSAGE;
  }
  ;
```

571. ⟨command⟩ ⟶ ⟨message or errmessage⟩**.** `STRING`. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  command:␣message_or_errmessage␣STRING
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Rule␣`command:␣message_or_errmessage␣STRING'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    if ($1 ≡ ERRMESSAGE) temp_strm << "ERROR!␣␣";
    temp_strm << $2 << endl;
    if ($1 ≡ ERRMESSAGE) temp_strm << "Type␣<RETURN>␣to␣continue:␣";
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    scanner_node→log_strm << temp_strm.str();
    temp_strm.str("");
    if ($1 ≡ ERRMESSAGE) getchar();
    cerr_mutex.Unlock();
  }
  ;
```

572. ⟨command⟩ ⟶ PAUSE. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  command: PAUSE
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Rule 'command: PAUSE'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    temp_strm << "Enter <RETURN> to continue: ";
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
    getchar();
    cerr_mutex.Unlock();
  }
  ;
```

573. ⟨command⟩ ⟶ `SHOW` ⟨query variable⟩. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  command:␣SHOW␣query_variable
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "Rule␣‘command:␣SHOW␣query_variable’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    Id_Node curr_id_node = static_cast⟨Id_Node⟩( $2 );
    if (curr_id_node ≡ 0) {
      temp_strm ≪ "WARNING!␣␣In␣‘yyparse’,␣rule␣‘command:␣SHOW␣query_variable’:" ≪ endl ≪
          "‘query_variable’␣is␣NULL.␣␣Can’t␣show.␣␣Continuing.";
      cerr_mutex.Lock( );
      cerr ≪ temp_strm.str( );
      cerr_mutex.Unlock( );
      scanner_node→log_strm ≪ temp_strm.str( );
      temp_strm.str("");
      $$ = 1;
    }    /* if (curr_id_node ≡ 0) */
    else if (curr_id_node→value ≡ 0) {
      temp_strm ≪ "WARNING!␣␣In␣‘yyparse’,␣rule␣‘command:␣SHOW␣query_variable’:" ≪
          endl ≪ "The␣‘value’␣element␣of␣‘query_variable’␣is␣NULL.␣␣" ≪
          "Can’t␣show.␣␣Continuing.";
      cerr_mutex.Lock( );
      cerr ≪ temp_strm.str( );
      cerr_mutex.Unlock( );
      scanner_node→log_strm ≪ temp_strm.str( );
      temp_strm.str("");
      $$ = 1;
    }    /* if (curr_id_node ≡ 0) */
    else if (curr_id_node ∧ curr_id_node→value) {
      Query_Node curr_query_node = static_cast⟨Query_Node⟩(curr_id_node→value);
      cerr_mutex.Lock( );
      curr_query_node→show("Query_Type:");
      cerr_mutex.Unlock( );
      $$ = 0;
```

```
    }
#undef DEBUG_OUTPUT
  }
  ;
```

574. ⟨command⟩ ⟶ `SHOW` ⟨datasource variable⟩. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  command: SHOW datasource_variable
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "Rule 'command: SHOW datasource_variable'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node curr_id_node = static_cast⟨Id_Node⟩($2);
    if (curr_id_node ≡ 0) {
      temp_strm ≪ "WARNING!  In 'yyparse', rule 'command: SHOW datasource_variable':" ≪
          endl ≪ "'datasource_variable' is NULL.  Can't show.  Continuing.";
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
      $$ = 1;
    }   /* if (curr_id_node ≡ 0) */
    else if (curr_id_node→value ≡ 0) {
      temp_strm ≪ "WARNING!  In 'yyparse', rule 'command: SHOW datasource_variable':" ≪
          endl ≪ "The 'value' element of 'datasource_variable' is NULL.  " ≪
          "Can't show.  Continuing.";
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
      $$ = 1;
    }   /* if (curr_id_node ≡ 0) */
    else if (curr_id_node ∧ curr_id_node→value) {
      Datasource_Node curr_datasource_node = static_cast⟨Datasource_Node⟩(curr_id_node→value);
      cerr_mutex.Lock();
      curr_datasource_node→show("Datasource_Type:");
      cerr_mutex.Unlock();
      $$ = 0;
```

```
    }
#undef DEBUG_OUTPUT
  }
  ;
```

575. ⟨command⟩ ⟶ `SHOW` ⟨datetime variable⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
command: SHOW datetime_variable
{
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
  Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
#ifdef DEBUG_OUTPUT
  temp_strm << endl << "Rule `command: SHOW datetime_variable'." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node->log_strm << temp_strm.str();
  temp_strm.str("");
#endif
  Id_Node curr_id_node = static_cast<Id_Node>($2);
  if (curr_id_node == 0) {
    temp_strm << "WARNING!  In `yyparse', rule `command: SHOW datetime_variable':" <<
        endl << "`datetime_variable' is NULL.  Can't show.  Continuing.";
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
    $$ = 1;
  } /* if (curr_id_node == 0) */
  else if (curr_id_node->value == 0) {
    temp_strm << "WARNING!  In `yyparse', rule `command: SHOW datetime_variable':" <<
        endl << "The `value' element of `datetime_variable' is NULL.  " <<
        "Can't show.  Continuing.";
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
    $$ = 1;
  } /* if (curr_id_node == 0) */
  else if (curr_id_node && curr_id_node->value) {
    Date_Time_Node curr_date_time_node = static_cast<Date_Time_Node>(curr_id_node->value);
    cerr_mutex.Lock();
    curr_date_time_node->show("Date_Time_Type:");
    cerr_mutex.Unlock();
    $$ = 0;
```

```
    }
#undef DEBUG_OUTPUT
  }
  ;
```

576. ⟨command⟩ ⟶ `SHOW` ⟨string expression⟩. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  command: SHOW string_expression
  {
#if 1      /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Rule 'command: SHOW string_expression'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    temp_strm << "string == " << $2 << endl;
#undef DEBUG_OUTPUT
  }
  ;
```

577. ⟨command⟩ ⟶ `OUTPUT TEX` ⟨string expression⟩. [LDF 2006.11.23.]

Log

[LDF 2006.11.23.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  command: OUTPUT TEX string_expression
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Rule 'command: OUTPUT TEX string_expression'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    stringstream temp_filename;
    tex_mutex.Lock();
    temp_filename << tex_filename_str << tex_file_ctr++ << ".tex";
    tex_file_strm.open(temp_filename.str().c_str());
    tex_file_strm << copyright_tex_str << endl;
    time_t tp;
    time(&tp);
    time_mutex.Lock();
    tm *local_time = localtime(&tp);
    char *datestamp = asctime(local_time);
    time_mutex.Unlock();
    tex_file_strm << "%% This file was generated by 'Scantest' on " << datestamp << "\n\n";
    tex_file_strm << "\\input chrtbase.tex\n\n" << $3 << "\n\n\\bye\n";
    tex_file_strm.close();
    tex_mutex.Unlock();
#undef DEBUG_OUTPUT
  }
  ;
```

578. Query expressions (`queryexp.w`). [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this file.

579.

⟨ queryexp.w 579 ⟩ ≡

```
static char queryexp_id_string[] = "$Id:␣queryexp.w,v␣1.3␣2007/01/02␣12:38:44␣lfinsto1␣Exp␣$";
```

This code is cited in sections 7 and 8.

This code is used in section 642.

580. query primary. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type␣<pointer_value>␣query_primary
```

581. ⟨query primary⟩ ⟶ ⟨query variable⟩. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  query_primary: query_variable
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `query_primary: query_variable'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node curr_id_node = static_cast<Id_Node>($1);
    if (curr_id_node && curr_id_node->value) {
      Query_Node curr_query_node = new Query_Type;
      *curr_query_node = *static_cast<Query_Node>(curr_id_node->value);
      $$ = static_cast<void *>(curr_query_node);
    }
    else $$ = 0;
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Exiting rule `query_primary: query_variable'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
  }
  ;
```

582. ⟨query primary⟩ ⟶ (⟨query expression⟩). [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  query_primary: OPEN_PARENTHESIS query_expression CLOSE_PARENTHESIS
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm << endl << "Rule 'query_primary: OPEN_PARENTHESIS query_expression CLOS\
        E_PARENTHESIS'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Query_Node q = static_cast⟨Query_Node⟩($2);
    $$ = $2;
  }
  ;
```

583. ⟨query primary⟩ ⟶ STRING. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  query_primary: STRING
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule ‘query_primary: STRING’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Query_Node curr_query_node = new Query_Type;
    curr_query_node→value_type = Query_Type::QUERY_TYPE_STRING_TYPE;
    curr_query_node→value = static_cast⟨void *⟩(new string);
    *static_cast⟨string *⟩(curr_query_node→value) = $1;
    $$ = static_cast⟨void *⟩(curr_query_node);
  }
  ;
```

584. ⟨query primary⟩ ⟶ INTEGER. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  query_primary: INTEGER
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule ‘query_primary: INTEGER’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Query_Node curr_query_node = new Query_Type;
    curr_query_node→value_type = Query_Type::INT_TYPE;
    curr_query_node→value = static_cast⟨void *⟩(new int);
    *static_cast⟨int *⟩(curr_query_node→value) = $1;
    $$ = static_cast⟨void *⟩(curr_query_node);
  }
  ;
```

585. ⟨query primary⟩ ⟶ `FLOAT`. [LDF 2006.12.14.]

Log

[LDF 2006.12.14.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
query_primary: FLOAT
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'query_primary: FLOAT'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Query_Node curr_query_node = new Query_Type;
    curr_query_node->value_type = Query_Type::FLOAT_TYPE;
    curr_query_node->value = static_cast<void *>(new float);
    *static_cast<float *>(curr_query_node->value) = $1;
    $$ = static_cast<void *>(curr_query_node);
  }
  ;
```

586. query secondary. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> query_secondary
```

587. ⟨query secondary⟩ ⟶ ⟨query primary⟩. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  query_secondary:␣query_primary
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘query_secondary:␣query_primary’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $1;
  }
  ;
```

588. ⟨query secondary⟩ ⟶ ⟨query secondary⟩ ⟨and or and not⟩ ⟨query primary⟩. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
query_secondary: query_secondary and_or_and_not query_primary
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `query_secondary: query_secondary and_or_and_not query\
        _primary'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Query_Node secondary = static_cast<Query_Node>($1);
    Query_Node primary = static_cast<Query_Node>($3);
    primary->up = secondary;
    primary->query_type = Query_Type::AND_TYPE;
    primary->negated = ($2 == AND_NOT) ? true : false;
    while (secondary->and_node != 0) secondary = secondary->and_node;
    secondary->and_node = primary;
    primary->query_ctr = secondary->query_ctr + 1;
    $$ = $1;
  }
  ;
```

589. query tertiary. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> query_tertiary
```

590. ⟨query tertiary⟩ ⟶ ⟨query secondary⟩. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  query_tertiary: query_secondary
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm << endl << "Rule `query_tertiary: query_secondary'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $1;
  }
  ;
```

591. ⟨query tertiary⟩ ⟶ ⟨query tertiary⟩ ⟨xor or xor not⟩ ⟨query secondary⟩. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
query_tertiary: query_tertiary xor_or_xor_not query_secondary
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `query_tertiary: query_tertiary xor_or_xor_not query_s\
        econdary'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Query_Node tertiary = static_cast<Query_Node>($1);
    Query_Node secondary = static_cast<Query_Node>($3);
    secondary->up = tertiary;
    secondary->query_type = Query_Type::XOR_TYPE;
    secondary->negated = ($2 == XOR_NOT) ? true : false;
    while (tertiary->xor_node != 0) tertiary = tertiary->xor_node;
    tertiary->xor_node = secondary;
    secondary->query_ctr = tertiary->query_ctr + 1;
    $$ = $1;
  }
  ;
```

592. query expression. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> query_expression
```

593. ⟨query expression⟩ ⟶ ⟨query tertiary⟩. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  query_expression:␣query_tertiary
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘query_expression:␣query_tertiary’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $1;
  }
  ;
```

594. ⟨query expression⟩ ⟶ ⟨query expression⟩ ⟨or or or not⟩ ⟨query tertiary⟩. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
query_expression:␣query_expression␣or_or_or_not␣query_tertiary
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘query_expression:␣query_expression␣or_or_or_not␣query\
        _tertiary’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Query_Node expression = static_cast⟨Query_Node⟩($1);
    Query_Node tertiary = static_cast⟨Query_Node⟩($3);
    tertiary→up = expression;
    tertiary→query_type = Query_Type::OR_TYPE;
    while (expression→or_node ≠ 0) expression = expression→or_node;
    tertiary→negated = ($2 ≡ OR_NOT) ? true : false;
    expression→or_node = tertiary;
    tertiary→query_ctr = expression→query_ctr + 1;
    $$ = $1;
  }
  ;
```

595. Boolean operators. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this section with the type declarations of *and_or_and_not*, *or_or_or_not*, and *xor_or_xor_not*.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type␣<int_value>␣and_or_and_not %type␣<int_value>␣or_or_or_not %type␣<int_value>␣xor_or_xor_no
```

596. ⟨and or and not⟩ ⟶ AND. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
and_or_and_not: AND
{
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'and_or_and_not: AND'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = AND;
}
;
```

597. ⟨and or and not⟩ ⟶ `AND_NOT`. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  and_or_and_not: AND_NOT
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `and_or_and_not: AND_NOT'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = AND_NOT;
  }
  ;
```

598. ⟨or or or not⟩ ⟶ `OR`. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  or_or_or_not:␣OR
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘or_or_or_not:␣OR’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = OR;
  }
  ;
```

599. ⟨or or or not⟩ ⟶ `OR_NOT`. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
or_or_or_not: OR_NOT
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `or_or_or_not: OR_NOT'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = OR_NOT;
  }
  ;
```

600. ⟨xor or xor not⟩ ⟶ `XOR`. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  xor_or_xor_not:␣XOR
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘xor_or_xor_not:␣XOR’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = XOR;
  }
  ;
```

601. ⟨xor or xor not⟩ ⟶ `XOR_NOT`. [LDF 2006.11.15.]

Log

[LDF 2006.11.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  xor_or_xor_not:␣XOR_NOT
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘xor_or_xor_not:␣XOR_NOT’." ≪ endl;
    cerr_mutex.Lock( );
    cerr ≪ temp_strm.str( );
    cerr_mutex.Unlock( );
    scanner_node→log_strm ≪ temp_strm.str( );
    temp_strm.str("");
#endif
    $$ = XOR_NOT;
  }
  ;
```

602. Datasource expressions (`dtsrcexp.w`). [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this file.

603.

```
⟨ dtsrcexp.w 603 ⟩ ≡
  static char dtsrcexp_id_string[ ] = "$Id:␣dtsrcexp.w,v␣1.3␣2007/01/02␣12:37:24␣lfinsto1␣Exp␣$";
```

This code is cited in sections 7 and 8.

This code is used in section 642.

604. datasource primary. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this type declaration.

```
⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
  %type␣<pointer_value>␣datasource_primary
```

605. ⟨datasource primary⟩ ⟶ ⟨datasource variable⟩. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datasource_primary: datasource_variable
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule 'datasource_primary: datasource_variable'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node curr_id_node = static_cast⟨Id_Node⟩($1);
    if (curr_id_node ∧ curr_id_node→value) {
      Datasource_Node curr_datasource_node = new Datasource_Type;
      *curr_datasource_node = *static_cast⟨Datasource_Node⟩(curr_id_node→value);
      $$ = static_cast⟨void *⟩(curr_datasource_node);
    }
    else $$ = 0;
#ifdef DEBUG_OUTPUT
    temp_strm ≪ endl ≪ "Exiting rule 'datasource_primary: datasource_variable'." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
  }
  ;
```

606. ⟨datasource primary⟩ ⟶ `(` ⟨datasource expression⟩ `).` [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datasource_primary: OPEN_PARENTHESIS datasource_expression CLOSE_PARENTHESIS
{
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `datasource_primary: OPEN_PARENTHESIS datasource_expre\
        ssion " << "CLOSE_PARENTHESIS'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Datasource_Node q = static_cast<Datasource_Node>($2);
    $$ = $2;
}
;
```

607. datasource secondary. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> datasource_secondary
```

608. ⟨datasource secondary⟩ ⟶ ⟨datasource primary⟩. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
   datasource_secondary: datasource_primary
   {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
      Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
      temp_strm << endl << "Rule `datasource_secondary: datasource_primary'." << endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node->log_strm << temp_strm.str();
      temp_strm.str("");
#endif
      $$ = $1;
   }
   ;
```

609. datasource tertiary. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this type declaration.

```
⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
   %type <pointer_value> datasource_tertiary
```

610. ⟨datasource tertiary⟩ ⟶ ⟨datasource secondary⟩. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datasource_tertiary: datasource_secondary
{
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `datasource_tertiary: datasource_secondary'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $1;
}
;
```

611. datasource expression. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> datasource_expression
```

612. ⟨datasource expression⟩ ⟶ ⟨datasource tertiary⟩. [LDF 2006.12.08.]

Log

[LDF 2006.12.08.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datasource_expression:␣datasource_tertiary
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datasource_expression:␣datasource_tertiary’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $1;
  }
  ;
```

613. String expressions (strings.w). [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this file.

```
⟨ strings.w 613 ⟩ ≡
  static char strings_id_string[] = "$Id:␣strings.w,v␣1.3␣2007/01/02␣12:38:44␣lfinsto1␣Exp␣$";
```

This code is used in section 642.

614. ⟨string primary⟩. [LDF 2006.11.16.]

Log

[2006.11.16.] Added this type declaration.

```
⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
  %type␣<string_value>␣string_primary
```

615. ⟨string primary⟩ ⟶ ⟨string variable⟩. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  string_primary: string_variable
  {
#if 1 /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'string_primary: string_variable'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node curr_id_node = static_cast<Id_Node>($1);
    if (curr_id_node == 0 || curr_id_node->value == 0) strcpy($$, "");
    else {
      strcpy($$, static_cast<string *>(curr_id_node->value)->c_str());
    }
  }
  ;
```

616. ⟨string primary⟩**:** `STRING`**.** [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  string_primary:␣STRING
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘string_primary:␣STRING’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    strcpy($$, $1);
  }
  ;
```

617. ⟨string primary⟩ ⟶ (⟨string expression⟩). [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
string_primary: OPEN_PARENTHESIS string_expression CLOSE_PARENTHESIS
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `string_primary: OPEN_PARENTHESIS string_expression CL\
        OSE_PARENTHESIS'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    strcpy($$, $2);
  }
  ;
```

618. string secondary. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <string_value> string_secondary
```

619. ⟨string secondary⟩ ⟶ ⟨string primary⟩. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  string_secondary:␣string_primary
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘string_secondary:␣string_primary’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    strcpy($$, $1);
  }
  ;
```

620. string tertiary. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this type declaration.

```
⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
  %type␣<string_value>␣string_tertiary
```

621. ⟨string tertiary⟩ ⟶ ⟨string secondary⟩. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this rule.

⟨ Parser rules 388 ⟩ +≡
```
string_tertiary: string_secondary
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `string_tertiary: string_secondary'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    strcpy($$, $1);
  }
  ;
```

622. **string expression.** [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
```
%type <string_value> string_expression
```

623. ⟨string expression⟩ ⟶ ⟨string tertiary⟩. [LDF 2006.11.16.]

Log

[LDF 2006.11.16.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
string_expression: string_tertiary
  {
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `string_expression: string_tertiary'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    strcpy($$, $1);
  }
  ;
```

624. Datetime expressions (dttmexp.w). [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this file.

625.

⟨ dttmexp.w 625 ⟩ ≡

```
static char dttmexp_id_string[] = "$Id: dttmexp.w,v 1.3 2007/01/02 12:37:35 lfinsto1 Exp $";
```

This code is cited in sections 7 and 8.

This code is used in section 642.

626. datetime primary. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> datetime_primary
```

627. ⟨datetime primary⟩ ⟶ ⟨datetime variable⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  datetime_primary: datetime_variable
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule 'datetime_primary: datetime_variable'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    Id_Node curr_id_node = static_cast<Id_Node>($1);
    if (curr_id_node && curr_id_node->value) {
      Date_Time_Node curr_datetime_node = new Date_Time_Type;
      *curr_datetime_node = *static_cast<Date_Time_Node>(curr_id_node->value);
      $$ = static_cast<void *>(curr_datetime_node);
    }
    else $$ = 0;
#ifdef DEBUG_OUTPUT
    temp_strm << endl << "Exiting rule 'datetime_primary: datetime_variable'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
  }
  ;
```

628. ⟨datetime primary⟩ ⟶ (⟨datetime expression⟩). [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_primary:␣OPEN_PARENTHESIS␣datetime_expression␣CLOSE_PARENTHESIS
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_primary:␣OPEN_PARENTHESIS␣" ≪
        "datetime_expression␣CLOSE_PARENTHESIS’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Date_Time_Node q = static_cast⟨Date_Time_Node⟩($2);
    $$ = $2;
  }
  ;
```

629. ⟨datetime primary⟩ ⟶ ⟨datetime element list⟩. [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_primary:␣datetime_element_list
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_primary:␣datetime_element_list." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    Date_Time_Node d = 0;
    datetime_assignment_func_1(scanner_node, d, ASSIGN, $1);
    $$ = static_cast⟨void *⟩(d);
  }
  ;
```

630. datetime secondary. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this type declaration.

```
⟨ Type declarations for non-terminal symbols 399 ⟩ +≡
  %type␣<pointer_value>␣datetime_secondary
```

631. ⟨datetime secondary⟩ ⟶ ⟨datetime primary⟩**.** [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datetime_secondary: datetime_primary
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `datetime_secondary: datetime_primary'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $1;
  }
  ;
```

632. datetime tertiary. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> datetime_tertiary
```

633. ⟨datetime tertiary⟩ ⟶ ⟨datetime secondary⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
  datetime_tertiary: datetime_secondary
  {
#if 1   /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `datetime_tertiary: datetime_secondary'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $1;
  }
  ;
```

634. datetime expression. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
  %type <pointer_value> datetime_expression
```

635. ⟨datetime expression⟩ ⟶ ⟨datetime tertiary⟩. [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datetime_expression: datetime_tertiary
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `datetime_expression: datetime_tertiary'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    $$ = $1;
  }
  ;
```

636. Datetime element list. [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this type declaration.

⟨ Type declarations for non-terminal symbols 399 ⟩ +≡

```
%type <pointer_value> datetime_element_list
```

637. ⟨datetime element list⟩ ⟶ `INTEGER COLON INTEGER`. [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_element_list:␣INTEGER␣COLON
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_element_list:␣INTEGER␣COLON␣INTEGER’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    vector⟨pair⟨int, void *⟩⟩ *v = new vector⟨pair⟨int, void *⟩⟩;
    int *i = new int($1);
    v→push_back(make_pair(INTEGER, static_cast⟨void *⟩(i)));
    $$ = static_cast⟨void *⟩(v);
  }
  ;
```

638. ⟨datetime element list⟩ ⟶ FLOAT COLON. [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this rule.

```
⟨ Parser rules 388 ⟩ +≡
  datetime_element_list:␣FLOAT␣COLON
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(parameter);
    temp_strm ≪ endl ≪ "Rule␣‘datetime_element_list:␣FLOAT␣COLON’." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    vector⟨pair⟨int, void *⟩⟩ *v = new vector⟨pair⟨int, void *⟩⟩;
    float *f = new float($1);
    v→push_back(make_pair(FLOAT, static_cast⟨void *⟩(f)));
    $$ = static_cast⟨void *⟩(v);
  }
  ;
```

639. ⟨datetime element list⟩ ⟶ ⟨datetime element list⟩ COLON INTEGER. [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datetime_element_list: datetime_element_list INTEGER COLON
{
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `datetime_element_list: datetime_element_list "
        "INTEGER COLON'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    vector<pair<int, void *>> *v = static_cast<vector<pair<int, void *>> *>($1);
    int *i = new int($2);
    v->push_back(make_pair(INTEGER, static_cast<void *>(i)));
    $$ = static_cast<void *>(v);
}
;
```

640. ⟨datetime element list⟩ ⟶ ⟨datetime element list⟩ `FLOAT COLON.` [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this rule.

⟨ Parser rules 388 ⟩ +≡

```
datetime_element_list: datetime_element_list FLOAT COLON
  {
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
    Scanner_Node scanner_node = static_cast<Scanner_Node>(parameter);
    temp_strm << endl << "Rule `datetime_element_list: datetime_element_list " <<
        "FLOAT COLON'." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    vector<pair<int, void *>> *v = static_cast<vector<pair<int, void *>> *>($1);
    float *f = new float($2);
    v->push_back(make_pair(FLOAT, static_cast<void *>(f)));
    $$ = static_cast<void *>(v);
  }
  ;
```

641. Putting the parser together.

642. Filename sections. These sections contain the CVS ID strings for the individual parser files. [LDF 2006.10.31.]

Log

[2006.10.31.] Added this section.

⟨ Filename sections 642 ⟩ ≡
⟨ `parser.w` 359 ⟩
⟨ `declrtns.w` 401 ⟩
⟨ `grpstmnt.w` 398 ⟩
⟨ `variabls.w` 422 ⟩
⟨ `assign.w` 446 ⟩
⟨ `queryexp.w` 579 ⟩
⟨ `dtsrcexp.w` 603 ⟩
⟨ `commands.w` 565 ⟩
⟨ `strings.w` 613 ⟩
⟨ `dttmexp.w` 625 ⟩

This code is used in sections 643 and 644.

643. This is what's written to `parser.yyy`. [LDF 2006.10.20.]

⟨ `parser.yyy` 643 ⟩ ≡
 $\boxed{\texttt{\%\{}}$
 ⟨ Include files 10 ⟩
 ⟨ Filename sections 642 ⟩
 ⟨ Declare *yylex* 301 ⟩
 ⟨ Declare *yyerror* 362 ⟩
#define `YYPARSE_PARAM` *parameter*
#define `YYLEX_PARAM` *parameter*
 stringstream *temp_strm*;
 $\boxed{\texttt{\%\}}}$
 ⟨ **union** declaration for `YYSTYPE` 363 ⟩
 ⟨ Bison declarations 364 ⟩
 ⟨ Token and type declarations 366 ⟩
 ⟨ Type declarations for non-terminal symbols 399 ⟩
 $\boxed{\texttt{\%\%}}$
 ⟨ Parser rules 388 ⟩

644. This is what *isn't* compiled. The file `parser.c` isn't compiled, but including ⟨ `parser.w` 359 ⟩ here prevents CWEAVE from issuing a warning. [LDF 2006.10.24.]

```
  ⟨ Filename sections 642 ⟩
#if 0      /* 1 */
  ⟨ Garbage 393 ⟩
#endif
```

645. Parser functions (`prsrfncs.web`). [LDF 2006.10.31.]

Log

[LDF 2006.10.31.] Created this file.

⟨ `prsrfncs.web` 645 ⟩ ≡

```
  static char id_string[] = "$Id:␣prsrfncs.web,v␣1.4␣2007/01/02␣12:20:31␣lfinsto1␣Exp␣$";
```

This code is used in section 695.

646. Include files.

⟨ Include files 10 ⟩ +≡

```
#include "localldf.h"
#ifdef NON_WIN_LDF
#include "nonwin.h"
#else
#include "stdafx.h"
#endif
#include "parser.hxx"
#include "dttmtype.h"
#include "idtype.h"
#include "scnrtype.h"
#include "scanner.h"
#include "querytyp.h"
#include "dtsrctyp.h"
```

647. Preprocessor macro calls.

⟨ Preprocessor macro calls 19 ⟩ +≡

```
#ifdef WIN_LDF
#pragma once
#endif
```

648. Parser function definitions. [LDF 2006.10.31.]

649. Functions for variables. [LDF 2006.11.02.]

Log

[2006.11.02.] Added this section.

---

650. *variable_func.* [LDF 2006.11.02.]

Log

[2006.11.02.] Added this function.

```
⟨ Define parser functions 650 ⟩ ≡
  void *Scan_Parse::variable_func(void *v, char *name){
#if 0     /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
      stringstream temp_strm;
      Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(v);
#ifdef DEBUG_OUTPUT
      temp_strm << "Entering␣‘Scan_Parse::variable_func’." << endl << "‘name’␣==␣" << name <<
          endl;
      cerr_mutex.Lock( );
      cerr << temp_strm.str( );
      cerr_mutex.Unlock( );
      if (scanner_node ≠ 0) scanner_node→log_strm << temp_strm.str( );
      temp_strm.str("");
#endif
```

See also sections 651, 652, 653, 654, 655, 656, 658, 659, 660, 662, 663, 664, 665, 666, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, and 693.

This code is used in section 695.

651. Error handling: *scanner_node* ≡ 0. This shouldn't ever happen. If it does, we can't continue. [LDF 2006.11.01.]

```
⟨ Define parser functions 650 ⟩ +≡
  if (scanner_node ≡ 0) {
    temp_strm << "ERROR!␣␣In␣‘Scan_Parse::variable_func’:" <<
        "‘scanner_node’␣==␣0.␣␣Can’t␣continue." << endl <<
        "Exiting␣function␣unsuccessfully␣with␣return␣value␣0." << endl;
    cerr_mutex.Lock( );
    cerr << temp_strm.str( );
    cerr_mutex.Unlock( );
    temp_strm.str("");
    return 0;
  }    /* if (scanner_node ≡ 0) */
  map⟨string, Id_Node⟩::iterator iter = scanner_node→id_map.find(name);
```

652. Error handling: *name* not found in *scanner_node→id_map*. [LDF 2006.11.03.]

⟨ Define parser functions 650 ⟩ +≡
```
if (iter ≡ scanner_node→id_map.end()) {
   temp_strm ≪ "ERROR!␣␣In␣`Scan_Parse::variable_func':" ≪ endl ≪
       "`" ≪ name ≪ "'␣not␣found␣in␣`scanner_node->id_map'." ≪ endl ≪
       "Exiting␣function␣unsuccessfully␣with␣return␣value␣0." ≪ endl;
   cerr_mutex.Lock();
   cerr ≪ temp_strm.str();
   cerr_mutex.Unlock();
   if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str();
   temp_strm.str("");
   scanner_node→float_vector.clear();
   return 0;
}    /* if (iter ≡ scanner_node→id_map.end()) */
```

653. *name* found in *scanner_node→id_map*. [LDF 2006.11.03.]

⟨ Define parser functions 650 ⟩ +≡
```
else    /* iter ≠ scanner_node→id_map.end() */
{ Id_Node curr_id_node = 0;
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "In␣`Scan_Parse::variable_func':" ≪ endl ≪ "`" ≪ name ≪
      "'␣found␣in␣`scanner_node->id_map'." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
```

654. *name* doesn't refer to an array. [LDF 2006.11.03.]

⟨ Define parser functions 650 ⟩ +≡
```
if (strchr(name, '¥') ≡ 0) {
   curr_id_node = iter→second;
}    /* if (strchr(name, '¥') ≡ 0) */
```

655. *name* refers to an array. [LDF 2006.11.03.]

```
⟨ Define parser functions 650 ⟩ +≡
  else    /* (strchr(name, '¥') ≠ 0) */
  {
#ifdef DEBUG_OUTPUT
    temp_strm ≪ "‘name’␣(‘" ≪ name ≪ "’)␣contains␣" ≪ scanner_node→float_vector.size() ≪
        "␣subscript␣placeholder";
    if (scanner_node→float_vector.size() > 1) temp_strm ≪ "s";
    temp_strm ≪ "." ≪ endl;
#endif
    int i = 0;
    for (vector⟨float⟩::const_iterator subscript_iter = scanner_node→float_vector.begin();
        subscript_iter ≠ scanner_node→float_vector.end(); ++subscript_iter) {
#ifdef DEBUG_OUTPUT
      temp_strm ≪ "Subscript␣" ≪ i++ ≪ "␣==␣" ≪ *subscript_iter ≪ endl;
#endif
    }    /* for */
#ifdef DEBUG_OUTPUT
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
#endif
    curr_id_node = scanner_node→lookup(name, true, iter→second→type);
  }    /* else (strchr(name, '¥') ≠ 0) */
```

656. Push a **Token_Type** object onto *scanner_node→token_stack*. [LDF 2006.11.03.]

```
⟨ Define parser functions 650 ⟩ +≡
  YYSTYPE curr_value;
  curr_value.pointer_value = static_cast⟨void *⟩(curr_id_node);
  scanner_node→token_stack.push(Token_Type(curr_id_node→type, curr_value));
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "Exiting␣‘Scan_Parse::variable_func’␣successfully␣" ≪
      "with␣return␣value␣‘curr_id_node’␣(cast␣to␣‘void*’)." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
  scanner_node→float_vector.clear();
  return static_cast⟨void *⟩(curr_id_node); }    /* else (iter ≠ scanner_node→id_map.end()) */
#undef DEBUG_OUTPUT
  }    /* End of Scan_Parse::variable_func definition. */
```

657. Functions for declarations. [LDF 2006.11.01.]

Log

[LDF 2006.11.01.] Added this section.

658. Declare variable function. [LDF 2006.11.01.]

Log

[2006.11.01.] Added this function.

[LDF 2006.11.01.] Added the **char** ∗*name* argument.

[LDF 2006.11.02.] Changed the return value to **Id_Node**. Now returning *curr_id_node* if successful, or 0 if not.

⟨ Define parser functions 650 ⟩ +≡

```
  Id_Node Scan_Parse::declare_variable_func(void *v, const unsigned short type, char *name){
#if 0     /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
      stringstream temp_strm;
      Scanner_Node scanner_node = static_cast<Scanner_Node>(v);
#ifdef DEBUG_OUTPUT
      temp_strm << "Entering `Scan_Parse::declare_variable_func'." << endl << "`type' == `" <<
          token_map[type] << "'" << endl << "`name' == " << name << endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      if (scanner_node != 0) scanner_node->log_strm << temp_strm.str();
      temp_strm.str("");
#endif
```

659. Error handling: *scanner_node* ≡ 0. This shouldn't ever happen. If it does, we can't continue. [LDF 2006.11.01.]

⟨ Define parser functions 650 ⟩ +≡

```
  if (scanner_node == 0) {
    temp_strm << "ERROR!  In `Scan_Parse::declare_variable_func':" <<
        "`scanner_node' == 0.  Can't continue." << endl <<
        "Exiting function unsuccessfully with return value 0." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    temp_strm.str("");
    return 0;
  }    /* if (scanner_node == 0) */
```

660.

Log

[LDF 2006.11.02.] Now setting *curr_id_node→scanner_node* = *scanner_node*.

```
⟨ Define parser functions 650 ⟩ +≡
  Id_Node curr_id_node = new Id_Type;
  curr_id_node→name = name;
  curr_id_node→type = type;
  curr_id_node→scanner_node = scanner_node;
  scanner_node→id_map[curr_id_node→name] = curr_id_node;
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "‘scanner_node->id_map[curr_id_node->name]->name’␣==␣" ≪
      scanner_node→id_map[curr_id_node→name]→name ≪ endl ≪
      "‘scanner_node->id_map[curr_id_node->name]->type’␣==␣‘" ≪
      token_map[scanner_node→id_map[curr_id_node→name]→type] ≪ "’" ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "Exiting␣‘Scan_Parse::declare_variable_func’." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
  return curr_id_node;
#undef DEBUG_OUTPUT
  }    /* End of Scan_Parse::declare_variable_func definition. */
```

661. Functions for assignments. [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this section.

---

662. *query_assignment_func_0.* [LDF 2006.11.02.]

Log

[LDF 2006.11.02.] Added this function.

[LDF 2006.11.13.] Added the arguments **unsigned int** *arg_0*, **unsigned int** *arg_1*, and **void** **value*.

```
⟨ Define parser functions 650 ⟩ +≡
  void *Scan_Parse::query_assignment_func_0(void *v, void *object, unsigned int
        assignment_type, unsigned int arg_0, unsigned int arg_1, void *value, bool negate){
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
      stringstream temp_strm;
      Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(v);
#ifdef DEBUG_OUTPUT
      temp_strm ≪ "Entering␣‘Scan_Parse::query_assignment_func_0’." ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
```

663. Error handling: *scanner_node* ≡ 0. This shouldn't ever happen. If it does, we can't continue. [LDF 2006.11.01.]

```
⟨ Define parser functions 650 ⟩ +≡
  if (scanner_node ≡ 0) {
    temp_strm ≪ "ERROR!␣␣In␣‘Scan_Parse::query_assignment_func_0’:" ≪
        "‘scanner_node’␣==␣0.␣␣Can’t␣continue." ≪ endl ≪
        "Exiting␣function␣unsuccessfully␣with␣return␣value␣0." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    temp_strm.str("");
    return 0;
  }    /* if (scanner_node ≡ 0) */
```

664.

⟨ Define parser functions 650 ⟩ +≡

```
if (object ≡ 0) {
    temp_strm ≪ "ERROR!␣␣In␣‘Scan_Parse::query_assignment_func_0’:" ≪
        endl ≪ "‘object’␣==␣0.␣␣Can’t␣perform␣assignment." ≪ endl ≪
        "Exiting␣function␣unsuccessfully␣with␣return␣value␣0." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    return 0;
}   /* if (object ≡ 0) */
```

665.

⟨ Define parser functions 650 ⟩ +≡

```
Id_Node curr_id_node = static_cast⟨Id_Node⟩(object);
Query_Node curr_query_node = 0;
if (curr_id_node→value ≡ 0) {
    curr_query_node = new Query_Type;
    curr_query_node→query_type = Query_Type::TOP_TYPE;
    curr_query_node→id_node = curr_id_node;
    curr_query_node→name = curr_id_node→name;
    curr_query_node→scanner_node = scanner_node;
    curr_id_node→value = static_cast⟨void *⟩(curr_query_node);
}   /* if (curr_id_node→value ≡ 0) */
else {
    curr_query_node = static_cast⟨Query_Node⟩(curr_id_node→value);
}
#if 0   /* 1 */
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "arg_0␣==␣" ≪ token_map[arg_0] ≪ endl ≪ "arg_1␣==␣" ≪ token_map[arg_1] ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
#endif
```

666. Targets. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this section.

⟨ Define parser functions 650 ⟩ +≡

```
if (arg_0 ≡ LOCAL) {
  if (arg_1 ≡ DATABASE ∧ find(curr_query_node→target_types.begin(), curr_query_node→target_types.end(),
        Query_Type::LOCAL_DATABASE_TARGET) ≡ curr_query_node→target_types.end())
    curr_query_node→target_types.push_back(Query_Type::LOCAL_DATABASE_TARGET);
  else if (arg_1 ≡ SERVER ∧ find(curr_query_node→target_types.begin(), curr_query_node→target_types.end(),
        Query_Type::LOCAL_SERVER_TARGET) ≡ curr_query_node→target_types.end())
    curr_query_node→target_types.push_back(Query_Type::LOCAL_SERVER_TARGET);
}   /* if (arg_0 ≡ LOCAL) */
else if (arg_0 ≡ REMOTE) {
  if (arg_1 ≡ DATABASE ∧ find(curr_query_node→target_types.begin(), curr_query_node→target_types.end(),
        Query_Type::REMOTE_DATABASE_TARGET) ≡ curr_query_node→target_types.end())
    curr_query_node→target_types.push_back(Query_Type::REMOTE_DATABASE_TARGET);
  else if (arg_1 ≡ SERVER ∧ find(curr_query_node→target_types.begin(), curr_query_node→target_types.end(),
        Query_Type::REMOTE_SERVER_TARGET) ≡ curr_query_node→target_types.end())
    curr_query_node→target_types.push_back(Query_Type::REMOTE_SERVER_TARGET);
}   /* else if (arg_0 ≡ REMOTE) */
```

667. Fields. [LDF 2006.11.13.]

Log

[2006.11.13.] Added this section.

---

668. Fields. [LDF 2006.11.14.]

Log

[LDF 2006.11.14.] Added this section.

[LDF 2006.12.05.] Added code for CREATOR.

[LDF 2006.12.12.] Added code for MAIN_CANONICAL_TITLE.

[LDF 2006.12.12.] Added code to account for tokens that refer to additional tables from the OAI database (dc_test).

[LDF 2006.12.12.] Added code to account for the tokens that refer to additional tables in the PICA database (PICA_DB).

```
⟨ Define parser functions 650 ⟩ +≡
  if (arg_0 ≡ ACCESS_NUMBER ∨ arg_0 ≡ AUTHOR ∨ arg_0 ≡ BIBLIOGRAPHIC_TYPE ∨ arg_0 ≡
      CALL_NUMBER ∨ arg_0 ≡ CLASSIFICATION ∨ arg_0 ≡ COMPANY ∨ arg_0 ≡ CONTENT_SUMMARY ∨ arg_0 ≡
      CONTRIBUTOR ∨ arg_0 ≡ CREATOR ∨ arg_0 ≡ DATABASE_PROVIDER ∨ arg_0 ≡ DESCRIPTION ∨ arg_0 ≡
      EXEMPLAR_PRODUCTION_NUMBER ∨ arg_0 ≡ IDENTIFIER ∨ arg_0 ≡ INSTITUTION ∨ arg_0 ≡
      LANGUAGE ∨ arg_0 ≡ MAIN_CANONICAL_TITLE ∨ arg_0 ≡ PERMUTATION_PATTERN ∨ arg_0 ≡
      PERSON ∨ arg_0 ≡ PHYSICAL_DESCRIPTION ∨ arg_0 ≡ PUBLISHER ∨ arg_0 ≡ RECORD ∨ arg_0 ≡
      REMOTE_ACCESS ∨ arg_0 ≡ RIGHTS ∨ arg_0 ≡ SOURCE ∨ arg_0 ≡ SUBJECT ∨ arg_0 ≡
      SUPERORDINATE_ENTITIES ∨ arg_0 ≡ TITLE ∨ arg_0 ≡ TYPE) { unsigned int curr_field_type;
  if (arg_0 ≡ ACCESS_NUMBER) curr_field_type = Query_Type::ACCESS_NUMBER_FIELD;
  else if (arg_0 ≡ AUTHOR) curr_field_type = Query_Type::AUTHOR_FIELD;
  else if (arg_0 ≡ BIBLIOGRAPHIC_TYPE) curr_field_type = Query_Type::BIBLIOGRAPHIC_TYPE_FIELD;
  else if (arg_0 ≡ CALL_NUMBER) curr_field_type = Query_Type::CALL_NUMBER_FIELD;
  else if (arg_0 ≡ CLASSIFICATION) curr_field_type = Query_Type::CLASSIFICATION_FIELD;
  else if (arg_0 ≡ COMPANY) curr_field_type = Query_Type::COMPANY_FIELD;
  else if (arg_0 ≡ CONTENT_SUMMARY) curr_field_type = Query_Type::CONTENT_SUMMARY_FIELD;
  else if (arg_0 ≡ CONTRIBUTOR) curr_field_type = Query_Type::CONTRIBUTOR_FIELD;
  else if (arg_0 ≡ CREATOR) curr_field_type = Query_Type::CREATOR_FIELD;
  else if (arg_0 ≡ DATABASE_PROVIDER) curr_field_type = Query_Type::DATABASE_PROVIDER_FIELD;
  else if (arg_0 ≡ DESCRIPTION) curr_field_type = Query_Type::DESCRIPTION_FIELD;
  else if (arg_0 ≡ EXEMPLAR_PRODUCTION_NUMBER)
    curr_field_type = Query_Type::EXEMPLAR_PRODUCTION_NUMBER_FIELD;
  else if (arg_0 ≡ IDENTIFIER) curr_field_type = Query_Type::IDENTIFIER_FIELD;
  else if (arg_0 ≡ INSTITUTION) curr_field_type = Query_Type::INSTITUTION_FIELD;
  else if (arg_0 ≡ LANGUAGE) curr_field_type = Query_Type::LANGUAGE_FIELD;
  else if (arg_0 ≡ MAIN_CANONICAL_TITLE)
    curr_field_type = Query_Type::MAIN_CANONICAL_TITLE_FIELD;
  else if (arg_0 ≡ PERMUTATION_PATTERN)
    curr_field_type = Query_Type::PERMUTATION_PATTERN_FIELD;
  else if (arg_0 ≡ PERSON) curr_field_type = Query_Type::PERSON_FIELD;
  else if (arg_0 ≡ PHYSICAL_DESCRIPTION)
    curr_field_type = Query_Type::PHYSICAL_DESCRIPTION_FIELD;
  else if (arg_0 ≡ PUBLISHER) curr_field_type = Query_Type::PUBLISHER_FIELD;
  else if (arg_0 ≡ RECORD) curr_field_type = Query_Type::RECORD_FIELD;
  else if (arg_0 ≡ REMOTE_ACCESS) curr_field_type = Query_Type::REMOTE_ACCESS_FIELD;
  else if (arg_0 ≡ RIGHTS) curr_field_type = Query_Type::RIGHTS_FIELD;
  else if (arg_0 ≡ SOURCE) curr_field_type = Query_Type::SOURCE_FIELD;
  else if (arg_0 ≡ SUBJECT) curr_field_type = Query_Type::SUBJECT_FIELD;
```

```
else if (arg_0 ≡ SUPERORDINATE_ENTITIES)
   curr_field_type = Query_Type::SUPERORDINATE_ENTITIES_FIELD;
else if (arg_0 ≡ TITLE) curr_field_type = Query_Type::TITLE_FIELD;
else if (arg_0 ≡ TYPE) curr_field_type = Query_Type::TYPE_FIELD;
while (curr_query_node→query_type ≠ Query_Type::TOP_TYPE)
   curr_query_node = curr_query_node→up;
if (curr_query_node→field_type ≡ Query_Type::QUERY_TYPE_NULL_TYPE ∧ curr_query_node→value ≡ 0) {
   curr_query_node→field_type = curr_field_type;
   string *curr_string = new string;
   *curr_string = static_cast⟨char *⟩(value);
   curr_query_node→value = static_cast⟨void *⟩(curr_string);
   curr_query_node→value_type = Query_Type::QUERY_TYPE_STRING_TYPE;
   curr_query_node→negated = negate;
}
```

669.

Log

[2006.11.14.] Added this section.

⟨ Define parser functions 650 ⟩ +≡

```
else {
  if (assignment_type ≡ ASSIGN ∨ assignment_type ≡ PLUS_ASSIGN) {
    while (curr_query_node→and_node ≠ 0) curr_query_node = curr_query_node→and_node;
    curr_query_node→and_node = new Query_Type;
    curr_query_node→and_node→query_ctr = curr_query_node→query_ctr + 1;
    curr_query_node→and_node→id_node = curr_id_node;
    curr_query_node→and_node→name = curr_id_node→name;
  }
  else if (assignment_type ≡ OR_ASSIGN) {
    while (curr_query_node→or_node ≠ 0) curr_query_node = curr_query_node→or_node;
    curr_query_node→or_node = new Query_Type;
    curr_query_node→or_node→query_ctr = curr_query_node→query_ctr + 1;
    curr_query_node→or_node→id_node = curr_id_node;
    curr_query_node→or_node→name = curr_id_node→name;
  }
  else if (assignment_type ≡ XOR_ASSIGN) {
    while (curr_query_node→xor_node ≠ 0) curr_query_node = curr_query_node→xor_node;
    curr_query_node→xor_node = new Query_Type;
    curr_query_node→xor_node→query_ctr = curr_query_node→query_ctr + 1;
    curr_query_node→xor_node→id_node = curr_id_node;
    curr_query_node→xor_node→name = curr_id_node→name;
  }
  if (curr_query_node→query_type ≡ Query_Type::TOP_TYPE) {
    if (assignment_type ≡ ASSIGN ∨ assignment_type ≡ PLUS_ASSIGN) {
      curr_query_node→and_node→up = curr_query_node;
    }
    else if (assignment_type ≡ OR_ASSIGN) {
      curr_query_node→or_node→up = curr_query_node;
    }
    else if (assignment_type ≡ XOR_ASSIGN) {
      curr_query_node→xor_node→up = curr_query_node;
    }
  }  /* if */
  else {
    if (assignment_type ≡ ASSIGN ∨ assignment_type ≡ PLUS_ASSIGN) {
      curr_query_node→and_node→up = curr_query_node→up;
    }
    else if (assignment_type ≡ OR_ASSIGN) {
      curr_query_node→or_node→up = curr_query_node→up;
    }
    else if (assignment_type ≡ XOR_ASSIGN) {
      curr_query_node→xor_node→up = curr_query_node→up;
    }
  }  /* else */
  if (assignment_type ≡ ASSIGN ∨ assignment_type ≡ PLUS_ASSIGN) {
```

```
        curr_query_node = curr_query_node→and_node;
        curr_query_node→query_type = Query_Type::AND_TYPE;
        curr_query_node→field_type = curr_field_type;
      }
      else if (assignment_type ≡ OR_ASSIGN) {
        curr_query_node = curr_query_node→or_node;
        curr_query_node→query_type = Query_Type::OR_TYPE;
        curr_query_node→field_type = curr_field_type;
      }
      else if (assignment_type ≡ XOR_ASSIGN) {
        curr_query_node = curr_query_node→xor_node;
        curr_query_node→query_type = Query_Type::XOR_TYPE;
        curr_query_node→field_type = curr_field_type;
      }
      string *curr_string = new string;
      *curr_string = static_cast⟨char *⟩(value);
      curr_query_node→value = static_cast⟨void *⟩(curr_string);
      curr_query_node→value_type = Query_Type::QUERY_TYPE_STRING_TYPE;
      curr_query_node→negated = negate;
    }    /* else */
  }    /* if */
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "Exiting␣‘Scan_Parse::query_assignment_func_0’␣successfully␣" ≪ endl ≪
      "with␣return␣value␣‘curr_query_node’,␣cast␣to␣‘void*’." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
  return static_cast⟨void *⟩(curr_query_node); }
    /* End of Scan_Parse::query_assignment_func_0 definition. */
```

670. *query_assignment_func_1.* [LDF 2006.12.19.]

Log

[LDF 2006.12.19.] Added this function.

```
⟨ Define parser functions 650 ⟩ +≡
  int Scan_Parse::query_assignment_func_1(Scanner_Node scanner_node, Id_Node &curr_id_node, void
        *&field_specifier, int assignment_operator, int negation_optional, int match_term_optional, void
        *&v, int type){ stringstream temp_strm;
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
# # undef DEBUG_OUTPUT
#endif
#ifdef DEBUG_OUTPUT
      temp_strm << endl << "Entering␣‘Scan_Parse::query_assignment_func_1’." << endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm << temp_strm.str();
      temp_strm.str("");
#endif
      Query_Node curr_query = 0;
      Query_Node traverse_query = 0;
      Query_Node temp_query = 0;
      if (assignment_operator ≡ ASSIGN ∨ assignment_operator ≡ PLUS_ASSIGN ∨ assignment_operator ≡
            AND_ASSIGN) {
        if (curr_id_node ∧ curr_id_node→value) {
          curr_query = static_cast⟨Query_Node⟩(curr_id_node→value);
          while (curr_query→and_node ≠ 0) curr_query = curr_query→and_node;
          if (type ≡ QUERY_TYPE) curr_query→and_node = static_cast⟨Query_Node⟩(v);
          else if (type ≡ DATETIME_TYPE) {
            temp_query = new Query_Type;
            temp_query→value_type = Query_Type::DATE_TIME_TYPE;
            temp_query→value = v;
            curr_query→and_node = temp_query;
          }
          curr_query→and_node→query_ctr = curr_query→query_ctr + 1;
          traverse_query = curr_query→or_node;
          while (traverse_query→or_node) traverse_query = traverse_query→or_node;
          if (traverse_query) curr_query→and_node→query_ctr += traverse_query→query_ctr;
          traverse_query = curr_query→xor_node;
          while (traverse_query→xor_node) traverse_query = traverse_query→xor_node;
          if (traverse_query) curr_query→and_node→query_ctr += traverse_query→query_ctr;
          curr_query→and_node→query_type = Query_Type::AND_TYPE;
          curr_query→and_node→up = curr_query;
          curr_query = curr_query→and_node;
          curr_query→set_field_specifier(field_specifier, true, scanner_node);
          curr_query→negated = (negation_optional);
        }    /* if */
        else if (curr_id_node) {
```

```
        if (type ≡ QUERY_TYPE) curr_id_node→value = v;
        else if (type ≡ DATETIME_TYPE) {
          temp_query = new Query_Type;
          temp_query→value_type = Query_Type::DATE_TIME_TYPE;
          temp_query→value = v;
          curr_id_node→value = static_cast⟨void *⟩(temp_query);
        }
        static_cast⟨Query_Node⟩(curr_id_node→value)→id_node = curr_id_node;
        static_cast⟨Query_Node⟩(curr_id_node→value)→name = curr_id_node→name;
        static_cast⟨Query_Node⟩(curr_id_node→value)→query_type = Query_Type::TOP_TYPE;
        static_cast⟨Query_Node⟩(curr_id_node→value)→query_ctr = 0;
        static_cast⟨Query_Node⟩(curr_id_node→value)→set_field_specifier(field_specifier, true,
            scanner_node);
        static_cast⟨Query_Node⟩(curr_id_node→value)→negated = (negation_optional);
        curr_query = static_cast⟨Query_Node⟩(curr_id_node→value);
      }   /* else */
      else {
        if (type ≡ QUERY_TYPE) {
          delete static_cast⟨Query_Node⟩(v);
        }
        else if (type ≡ DATETIME_TYPE) {
          delete static_cast⟨Date_Time_Node⟩(v);
        }
        v = 0;
        curr_query = 0;
      }
    }   /* if (assignment_operator ≡ ASSIGN ∨ assignment_operator ≡
          PLUS_ASSIGN ∨ assignment_operator ≡ AND_ASSIGN) */
    else if (assignment_operator ≡ OR_ASSIGN) {
      if (curr_id_node ∧ curr_id_node→value) {
        curr_query = static_cast⟨Query_Node⟩(curr_id_node→value);
        while (curr_query→or_node ≠ 0) curr_query = curr_query→or_node;
        if (type ≡ QUERY_TYPE) curr_query→or_node = static_cast⟨Query_Node⟩(v);
        else if (type ≡ DATETIME_TYPE) {
          temp_query = new Query_Type;
          temp_query→value_type = Query_Type::DATE_TIME_TYPE;
          temp_query→value = v;
          curr_query→or_node = temp_query;
        }
        curr_query→or_node→query_type = Query_Type::OR_TYPE;
        curr_query→or_node→query_ctr = curr_query→query_ctr + 1;
        traverse_query = curr_query→and_node;
        while (traverse_query→and_node) traverse_query = traverse_query→and_node;
        if (traverse_query) curr_query→or_node→query_ctr += traverse_query→query_ctr;
        traverse_query = curr_query→xor_node;
        while (traverse_query→xor_node) traverse_query = traverse_query→xor_node;
        if (traverse_query) curr_query→or_node→query_ctr += traverse_query→query_ctr;
        curr_query→or_node→up = curr_query;
        curr_query = curr_query→or_node;
        curr_query→set_field_specifier(field_specifier, true, scanner_node);
        curr_query→negated = (negation_optional);
      }   /* if */
```

```
    else if (curr_id_node) {
      if (type ≡ QUERY_TYPE) curr_id_node→value = v;
      else if (type ≡ DATETIME_TYPE) {
        temp_query = new Query_Type;
        temp_query→value_type = Query_Type::DATE_TIME_TYPE;
        temp_query→value = v;
        curr_id_node→value = static_cast⟨void *⟩(temp_query);
      }
      static_cast⟨Query_Node⟩(curr_id_node→value)→id_node = curr_id_node;
      static_cast⟨Query_Node⟩(curr_id_node→value)→name = curr_id_node→name;
      static_cast⟨Query_Node⟩(curr_id_node→value)→query_type = Query_Type::TOP_TYPE;
      static_cast⟨Query_Node⟩(curr_id_node→value)→set_field_specifier(field_specifier, true,
          scanner_node);
      static_cast⟨Query_Node⟩(curr_id_node→value)→query_ctr = 0;
      static_cast⟨Query_Node⟩(curr_id_node→value)→negated = (negation_optional);
      curr_query = static_cast⟨Query_Node⟩(curr_id_node→value);
    }   /* else */
    else {
      if (type ≡ QUERY_TYPE) {
        delete static_cast⟨Query_Node⟩(v);
      }
      else if (type ≡ DATETIME_TYPE) {
        delete static_cast⟨Date_Time_Node⟩(v);
      }
      v = 0;
      curr_query = 0;
      return 1;
    }
  }   /* if (assignment_operator ≡ OR_ASSIGN) */
  else if (assignment_operator ≡ XOR_ASSIGN) {
    if (curr_id_node ∧ curr_id_node→value) {
      curr_query = static_cast⟨Query_Node⟩(curr_id_node→value);
      while (curr_query→xor_node ≠ 0) curr_query = curr_query→xor_node;
      if (type ≡ QUERY_TYPE) curr_query→xor_node = static_cast⟨Query_Node⟩(v);
      else if (type ≡ DATETIME_TYPE) {
        temp_query = new Query_Type;
        temp_query→value_type = Query_Type::DATE_TIME_TYPE;
        temp_query→value = v;
        curr_query→xor_node = temp_query;
      }
      curr_query→xor_node→query_ctr = curr_query→query_ctr + 1;
      traverse_query = curr_query→and_node;
      while (traverse_query→and_node) traverse_query = traverse_query→and_node;
      if (traverse_query) curr_query→xor_node→query_ctr += traverse_query→query_ctr;
      traverse_query = curr_query→or_node;
      while (traverse_query→or_node) traverse_query = traverse_query→or_node;
      if (traverse_query) curr_query→xor_node→query_ctr += traverse_query→query_ctr;
      curr_query→xor_node→query_type = Query_Type::XOR_TYPE;
      curr_query→xor_node→up = curr_query;
      curr_query = curr_query→xor_node;
      curr_query→set_field_specifier(field_specifier, true, scanner_node);
      curr_query→negated = (negation_optional);
```

```
      }    /* if */
      else if (curr_id_node) {
        if (type ≡ QUERY_TYPE) curr_id_node→value = v;
        else if (type ≡ DATETIME_TYPE) {
          temp_query = new Query_Type;
          temp_query→value_type = Query_Type::DATE_TIME_TYPE;
          temp_query→value = v;
          curr_id_node→value = static_cast⟨void *⟩(temp_query);
        }
        static_cast⟨Query_Node⟩(curr_id_node→value)→id_node = curr_id_node;
        static_cast⟨Query_Node⟩(curr_id_node→value)→name = curr_id_node→name;
        static_cast⟨Query_Node⟩(curr_id_node→value)→query_type = Query_Type::TOP_TYPE;
        static_cast⟨Query_Node⟩(curr_id_node→value)→query_ctr = 0;
        static_cast⟨Query_Node⟩(curr_id_node→value)→set_field_specifier(field_specifier, true,
            scanner_node);
        static_cast⟨Query_Node⟩(curr_id_node→value)→negated = (negation_optional);
        curr_query = static_cast⟨Query_Node⟩(curr_id_node→value);
      }    /* else */
      else {
        if (type ≡ QUERY_TYPE) {
          delete static_cast⟨Query_Node⟩(v);
        }
        else if (type ≡ DATETIME_TYPE) {
          delete static_cast⟨Date_Time_Node⟩(v);
        }
        v = 0;
        curr_query = 0;
        return 1;
      }
    }    /* if (assignment_operator ≡ XOR_ASSIGN) */
```

671.

⟨ Define parser functions 650 ⟩ +≡

```
  else {
    curr_query = 0;
    return 1;
  }
```

672. Set *match_value*. [LDF 2006.12.12.]

Log

[2006.12.12.] Added this section.

⟨ Define parser functions 650 ⟩ +≡

```
  if (curr_query) {
    curr_query→match_value = match_term_optional;
  }    /* if (curr_query) */
```

673.

```
⟨ Define parser functions 650 ⟩ +≡
#ifdef DEBUG_OUTPUT
  temp_strm ≪ endl ≪ "Exiting␣‘Scan_Parse::query_assignment_func_1’␣successfully␣" ≪
      "with␣return␣value␣0." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
#endif
  return 0;
#undef DEBUG_OUTPUT
  }     /* End of Scan_Parse::query_assignment_func_1 definition. */
```

674. *datetime_assignment_func_0.* [LDF 2006.12.15.]

Log

[LDF 2006.12.15.] Added this function.

To Do

[LDF 2006.12.18.] Add range checking and code for other assignment operators.

```
⟨ Define parser functions 650 ⟩ +≡
  int Scan_Parse::datetime_assignment_func_0(void *v, Date_Time_Node &curr_date_time_node, int
          specifier, int op, void *val, int type){
#if 1     /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
      stringstream temp_strm;
      Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(v);
#ifdef DEBUG_OUTPUT
      temp_strm ≪ "Entering␣‘Scan_Parse::datetime_assignment_func_0’." ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
#ifdef DEBUG_OUTPUT
      temp_strm ≪ "‘specifier’␣==␣" ≪ token_map[specifier] ≪ endl ≪ "‘op’␣==␣" ≪
          token_map[op] ≪ endl ≪ "‘type’␣==␣" ≪ token_map[type] ≪ endl;
      cerr_mutex.Lock();
      cerr ≪ temp_strm.str();
      cerr_mutex.Unlock();
      scanner_node→log_strm ≪ temp_strm.str();
      temp_strm.str("");
#endif
```

675. Error handling: *val* ≡ 0. [LDF 2006.12.18.]

⟨ Define parser functions 650 ⟩ +≡

```
if (val ≡ 0) {
  temp_strm ≪ "ERROR!␣␣In␣‘Scan_Parse::datetime_assignment_func_0’:" ≪
      endl ≪ "‘val’␣==␣0’.␣␣Can’t␣assign." ≪ endl ≪
      "Exiting␣function␣unsuccessfully␣with␣return␣value␣1." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
  return 1;
}    /* if (val ≡ 0) */
```

676.

⟨ Define parser functions 650 ⟩ +≡

```
if (curr_date_time_node ≡ 0) {
  curr_date_time_node = new Date_Time_Type;
}    /* if (curr_date_time_node ≡ 0) */
```

677.

```
⟨ Define parser functions 650 ⟩ +≡
  if (op ≡ ASSIGN) {
    if (specifier ≡ YEAR_RANGE_BEGIN) {
      if (curr_date_time_node→year_range_begin ≡ 0) curr_date_time_node→year_range_begin = new short;
      *curr_date_time_node→year_range_begin = *static_cast⟨short *⟩(val);
    }    /* if (specifier ≡ YEAR_RANGE_BEGIN) */
    if (specifier ≡ YEAR_RANGE_END) {
      if (curr_date_time_node→year_range_end ≡ 0) curr_date_time_node→year_range_end = new short;
      *curr_date_time_node→year_range_end = *static_cast⟨short *⟩(val);
    }    /* if (specifier ≡ YEAR_RANGE_END) */
    if (specifier ≡ YEAR) {
      if (curr_date_time_node→year ≡ 0) curr_date_time_node→year = new short;
      *curr_date_time_node→year = *static_cast⟨short *⟩(val);
    }    /* if (specifier ≡ YEAR) */
    else if (specifier ≡ MONTH) {
      if (curr_date_time_node→month ≡ 0) curr_date_time_node→month = new unsigned short;
      *curr_date_time_node→month = *static_cast⟨unsigned short *⟩(val);
    }    /* else if (specifier ≡ MONTH) */
    else if (specifier ≡ DAY) {
      if (curr_date_time_node→day ≡ 0) curr_date_time_node→day = new unsigned short;
      *curr_date_time_node→day = *static_cast⟨unsigned short *⟩(val);
    }    /* else if (specifier ≡ DAY) */
    else if (specifier ≡ HOUR) {
      if (curr_date_time_node→hour ≡ 0) curr_date_time_node→hour = new unsigned short;
      *curr_date_time_node→hour = *static_cast⟨unsigned short *⟩(val);
    }    /* else if (specifier ≡ HOUR) */
    else if (specifier ≡ MINUTE) {
      if (curr_date_time_node→minute ≡ 0) curr_date_time_node→minute = new unsigned short;
      *curr_date_time_node→minute = *static_cast⟨unsigned short *⟩(val);
    }    /* else if (specifier ≡ MINUTE) */
    else if (specifier ≡ SECOND) {
      if (curr_date_time_node→second ≡ 0) curr_date_time_node→second = new float;
      if (type ≡ FLOAT) *curr_date_time_node→second = *static_cast⟨float *⟩(val);
      else if (type ≡ INTEGER)
        *curr_date_time_node→second = static_cast⟨float⟩(*static_cast⟨int *⟩(val));
    }    /* else if (specifier ≡ SECOND) */
  }    /* if (op ≡ ASSIGN) */
```

678.

⟨ Define parser functions 650 ⟩ +≡

```
else
  if (op ≡ PLUS_ASSIGN) {
    if (specifier ≡ YEAR_RANGE_BEGIN) {
      if (curr_date_time_node→year_range_begin ≡ 0)
        curr_date_time_node→year_range_begin = new short(0);
      *curr_date_time_node→year_range_begin += *static_cast⟨short *⟩(val);
    }    /* if (specifier ≡ YEAR_RANGE_BEGIN) */
    if (specifier ≡ YEAR_RANGE_END) {
      if (curr_date_time_node→year_range_end ≡ 0)
        curr_date_time_node→year_range_end = new short(0);
      *curr_date_time_node→year_range_end += *static_cast⟨short *⟩(val);
    }    /* if (specifier ≡ YEAR_RANGE_END) */
    if (specifier ≡ YEAR) {
      if (curr_date_time_node→year ≡ 0) curr_date_time_node→year = new short(0);
      *curr_date_time_node→year += *static_cast⟨short *⟩(val);
    }    /* if (specifier ≡ YEAR) */
    else if (specifier ≡ MONTH) {
      if (curr_date_time_node→month ≡ 0) curr_date_time_node→month = new unsigned short(0);
      *curr_date_time_node→month += *static_cast⟨unsigned short *⟩(val);
    }    /* else if (specifier ≡ MONTH) */
    else if (specifier ≡ DAY) {
      if (curr_date_time_node→day ≡ 0) curr_date_time_node→day = new unsigned short(0);
      *curr_date_time_node→day += *static_cast⟨unsigned short *⟩(val);
    }    /* else if (specifier ≡ DAY) */
    else if (specifier ≡ HOUR) {
      if (curr_date_time_node→hour ≡ 0) curr_date_time_node→hour = new unsigned short(0);
      *curr_date_time_node→hour += *static_cast⟨unsigned short *⟩(val);
    }    /* else if (specifier ≡ HOUR) */
    else if (specifier ≡ MINUTE) {
      if (curr_date_time_node→minute ≡ 0) curr_date_time_node→minute = new unsigned short(0);
      *curr_date_time_node→minute += *static_cast⟨unsigned short *⟩(val);
    }    /* else if (specifier ≡ MINUTE) */
    else if (specifier ≡ SECOND) {
      if (curr_date_time_node→second ≡ 0) curr_date_time_node→second = new float(0);
      if (type ≡ FLOAT) *curr_date_time_node→second += *static_cast⟨float *⟩(val);
      else if (type ≡ INTEGER)
        *curr_date_time_node→second += static_cast⟨float⟩(*static_cast⟨int *⟩(val));
    }    /* else if (specifier ≡ SECOND) */
  }    /* else if (op ≡ PLUS_ASSIGN) */
```

679.

⟨ Define parser functions 650 ⟩ +≡

```
else
  if (op ≡ MINUS_ASSIGN) { }    /* else if (op ≡ MINUS_ASSIGN) */
```

680.

⟨ Define parser functions 650 ⟩ +≡

```
else
  if (op ≡ TIMES_ASSIGN) { }    /* else if (op ≡ TIMES_ASSIGN) */
```

681.

```
⟨ Define parser functions 650 ⟩ +≡
  else
    if (op ≡ DIVIDE_ASSIGN) { }     /* else if (op ≡ DIVIDE_ASSIGN) */
```

682. Error handling: *op* has invalid value. [LDF 2006.12.18.]

```
⟨ Define parser functions 650 ⟩ +≡
  else {
    temp_strm ≪ "ERROR!␣␣In␣‘Scan_Parse::datetime_assignment_func_0’:" ≪ endl ≪
        "‘op’␣has␣invalid␣value:␣␣" ≪ token_map[op] ≪ "␣(" ≪ op ≪ ")" ≪ endl ≪
        "Exiting␣function␣unsuccessfully␣with␣return␣value␣1." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    return 1;
  }     /* else */
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "Exiting␣‘Scan_Parse::datetime_assignment_func_0’␣" ≪
      "successfully␣with␣return␣value␣0." ≪ endl;
  cerr_mutex.Lock();
  cerr ≪ temp_strm.str();
  cerr_mutex.Unlock();
  if (scanner_node ≠ 0) scanner_node→log_strm ≪ temp_strm.str();
  temp_strm.str("");
  getchar();
#endif
  return 0; }     /* End of Scan_Parse::datetime_assignment_func_0 definition. */
```

683. *datetime_assignment_func_1.* [LDF 2006.12.18.]

Log

[LDF 2006.12.18.] Added this function.

To Do

[LDF 2006.12.18.] Add range checking and perhaps code for handling different assignment operators.

```
⟨ Define parser functions 650 ⟩ +≡
  int Scan_Parse::datetime_assignment_func_1(void *v, Date_Time_Node &d, int op, void *&w){
#if 1    /* 0 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
      stringstream temp_strm;
      Scanner_Node scanner_node = static_cast⟨Scanner_Node⟩(v);
#ifdef DEBUG_OUTPUT
      temp_strm << "Entering␣`Scan_Parse::datetime_assignment_func_1'." << endl;
      cerr_mutex.Lock();
      cerr << temp_strm.str();
      cerr_mutex.Unlock();
      if (scanner_node ≠ 0) scanner_node→log_strm << temp_strm.str();
      temp_strm.str("");
#endif
```

684.

```
⟨ Define parser functions 650 ⟩ +≡
  vector⟨pair⟨int, void *⟩⟩ *vec = static_cast⟨vector⟨pair⟨int, void *⟩⟩ *⟩(w);
  if (vec ≡ 0) {
    temp_strm << "ERROR!␣␣In␣`Scan_Parse::datetime_assignment_func_1':" <<
        endl << "`datetime_element_list'␣==␣NULL.␣␣" <<
        "Exiting␣function␣unsuccessfully␣with␣return␣value␣1." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm << temp_strm.str();
    temp_strm.str("");
    return 1;
  }   /* if (vec ≡ 0) */
```

685.

```
⟨ Define parser functions 650 ⟩ +≡
  if (vec→size() ≡ 0) {
    temp_strm ≪ "ERROR!␣␣In␣‘Scan_Parse::datetime_assignment_func_1’:" ≪ endl ≪
        "‘vec’␣is␣empty.␣␣" ≪ "Exiting␣function␣unsuccessfully␣with␣return␣value␣1." ≪
        endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    delete vec;
    vec = 0;
    return 1;
  }    /* if (vec→size() ≡ 0) */
```

686.

```
⟨ Define parser functions 650 ⟩ +≡
  if (vec→size() > 0 ∧ vec→at(0).first ≠ INTEGER    /* year */
   ∨ vec→size() > 1 ∧ vec→at(1).first ≠ INTEGER    /* month */
   ∨ vec→size() > 2 ∧ vec→at(2).first ≠ INTEGER    /* day */
   ∨ vec→size() > 3 ∧ vec→at(3).first ≠ INTEGER    /* hour */
   ∨ vec→size() > 4 ∧ vec→at(4).first ≠ INTEGER    /* minute */
  ) {
    temp_strm ≪ "ERROR!␣␣In␣‘Scan_Parse::datetime_assignment_func_1’:" ≪
        endl ≪ "Invalid␣value␣in␣‘vec’." ≪ endl ≪
        "Exiting␣function␣unsuccessfully␣with␣return␣value␣1." ≪ endl;
    cerr_mutex.Lock();
    cerr ≪ temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node→log_strm ≪ temp_strm.str();
    temp_strm.str("");
    for (vector⟨pair⟨int, void *⟩⟩::iterator iter = vec→begin(); iter ≠ vec→end(); ++iter) {
      if (iter→first ≡ INTEGER) {
        delete static_cast⟨int *⟩(iter→second);
      }
      else if (iter→first ≡ FLOAT) {
        delete static_cast⟨float *⟩(iter→second);
      }
      iter→second = 0;
    }    /* for */
    vec→clear();
    delete vec;
    vec = 0;
    return 1;
  }    /* if */
```

687. Year. [LDF 2006.12.18.]

⟨ Define parser functions 650 ⟩ +≡
  if $(d \equiv 0)$ $d =$ **new Date_Time_Type**;
  $d\text{→}year =$ **static_cast**⟨**short** $*$⟩$(vec\text{→}at(0).second)$;
  $vec\text{→}at(0).second = 0$;

688. Month. [LDF 2006.12.18.]

⟨ Define parser functions 650 ⟩ +≡
  if $(vec\text{→}size() > 1)$ {
    $d\text{→}month =$ **static_cast**⟨**unsigned short** $*$⟩$(vec\text{→}at(1).second)$;
    $vec\text{→}at(1).second = 0$;
  }

689. Day. [LDF 2006.12.18.]

⟨ Define parser functions 650 ⟩ +≡
  if $(vec\text{→}size() > 2)$ {
    $d\text{→}day =$ **static_cast**⟨**unsigned short** $*$⟩$(vec\text{→}at(2).second)$;
    $vec\text{→}at(2).second = 0$;
  }

690. Hour. [LDF 2006.12.18.]

⟨ Define parser functions 650 ⟩ +≡
  if $(vec\text{→}size() > 3)$ {
    $d\text{→}hour =$ **static_cast**⟨**unsigned short** $*$⟩$(vec\text{→}at(3).second)$;
    $vec\text{→}at(3).second = 0$;
  }

691. Minute. [LDF 2006.12.18.]

⟨ Define parser functions 650 ⟩ +≡
  if $(vec\text{→}size() > 4)$ {
    $d\text{→}minute =$ **static_cast**⟨**unsigned short** $*$⟩$(vec\text{→}at(4).second)$;
    $vec\text{→}at(4).second = 0$;
  }

692. Second. [LDF 2006.12.18.]

⟨ Define parser functions 650 ⟩ +≡
  if $(vec\text{→}size() > 5)$ {
    if $(vec\text{→}at(5).first \equiv$ `FLOAT`$)$ $d\text{→}second =$ **static_cast**⟨**float** $*$⟩$(vec\text{→}at(5).second)$;
    else {
      $d\text{→}second =$ **new float**;
      $*d\text{→}second =$ **static_cast**⟨**float**⟩$(*$**static_cast**⟨**int** $*$⟩$(vec\text{→}at(5).second))$;
      delete static_cast⟨**int** $*$⟩$(vec\text{→}at(5).second)$;
    }
    $vec\text{→}at(5).second = 0$;
  }

693.

⟨ Define parser functions 650 ⟩ +≡

```
#ifdef DEBUG_OUTPUT
  temp_strm << "Exiting␣'Scan_Parse::datetime_assignment_func_1'␣" <<
      "successfully␣with␣return␣value␣0." << endl;
  cerr_mutex.Lock();
  cerr << temp_strm.str();
  cerr_mutex.Unlock();
  if (scanner_node ≠ 0) scanner_node→log_strm << temp_strm.str();
  temp_strm.str("");
#endif
  return 0; }     /* End of Scan_Parse::datetime_assignment_func_1 definition. */
```

694. Putting Parser Functions together. [LDF 2006.10.31.]

695. This is what's compiled.

```
⟨ Include files 10 ⟩
⟨ prsrfncs.web 645 ⟩
using namespace std;
using namespace Scan_Parse;
⟨ Define parser functions 650 ⟩
```

696. Scan Test (scantest.web). [LDF 2006.10.17.]

This file is used with Microsoft Visual Studio. It's not used when using The GNU Compiler Collection (GCC). [LDF 2006.10.24.]

Test application for the scanner *yylex* for ZTest. [LDF 2006.10.17.]

Log

[LDF 2006.10.17.] Created this file.

```
⟨ scantest.web 696 ⟩ ≡
  static char id_string[] = "$Id: scantest.web,v 1.3 2007/01/02 09:39:20 lfinsto1 Exp $";
```

This code is cited in sections 6 and 8.

This code is used in section 700.

697. Include files. [LDF 2006.10.17.]

```
⟨ Include files 10 ⟩ +≡
#include "stdafx.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

698. Global variable declarations. [LDF 2006.10.19.]

Log

[LDF 2006.10.19.] Added this section.

```
⟨ Global variables 698 ⟩ ≡
  ofstream log_strm;
  CMutex log_strm_mutex;
  CMutex cerr_mutex;
  CMutex cout_mutex;
  ifstream in_strm;
```

See also section 703.

This code is used in sections 700 and 715.

699. Define _tmain. [LDF 2006.10.17.]

Log

[LDF 2006.10.17.] Added this function.

```
⟨ Define _tmain 699 ⟩ ≡
  CWinApp theApp;
```

```
using namespace std;
int _tmain(int argc, TCHAR * argv[], TCHAR * envp[])
{
  int nRetCode = 0;   /* MFC initialisieren und drucken. Bei Fehlschlag Fehlermeldung aufrufen. */
  if (¬AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0)) {
      /* TODO: Passen Sie den Fehlercode an Ihre Anforderungen an */
    _tprintf(_T("Schwerwiegender␣Fehler:␣MFC-Initialisierung␣fehlgeschlagen\n"));
    nRetCode = 1;
  }
  else {
#define DEBUG_OUPUT
    using namespace Scan_Parse;
    int status;
    log_strm_mutex.Lock();
    log_strm.open("log.txt");
    log_strm << "%%␣log.txt\n\n" << copyright_tex_str << endl;
    log_strm_mutex.Unlock();
    status = initialize_keyword_map();
    status = show_keyword_map();
    YYSTYPE * value = new YYSTYPE;
    YYLTYPE * location = 0;
    Scanner_Type *scanner_node = new Scanner_Type;
    strcpy(scanner_node→in_filename, "commands.txt");
    in_strm.open(scanner_node→in_filename);
    status = 0;
    while (status ≠ TERMINATE) status = yylex(value, location, static_cast⟨void *⟩(scanner_node));
#ifdef DEBUG_OUTPUT
    temp_strm << "In␣`t_main':␣␣Finished␣scanning." << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    log_strm_mutex.Lock();
    log_strm << temp_strm.str();
    log_strm_mutex.Unlock();
    temp_strm.str("");
#endif
    in_strm.close();
    log_strm_mutex.Lock();
    log_strm.close();
    log_strm_mutex.Unlock();
    delete scanner_node;
    scanner_node = 0;
    char c;
    cerr_mutex.Lock();
    cerr << "`yylex'␣returned␣" << status << endl << "Enter␣a␣character␣to␣exit:␣";
    cerr_mutex.Unlock();
    cin >> c;
    return 0;
#undef DEBUG_OUPUT
  }
```

```
    return nRetCode;
  }    /* End of _tmain definition. */
```

This code is used in section 700.

700. Putting Scan Test together. [LDF 2006.10.17.]

```
⟨ Include files 10 ⟩
⟨ scantest.web 696 ⟩
using namespace std;
⟨ Global variables 698 ⟩
⟨ Define _tmain 699 ⟩
```

701. Main when using GCC (main.web). [LDF 2006.10.20.]

Log

[LDF 2006.10.20.] Created this file.

```
⟨ main.web 701 ⟩ ≡
  static char id_string[] = "$Id: main.web,v 1.6 2007/01/02 12:45:46 lfinsto1 Exp $";
```

This code is cited in sections 6 and 8.

This code is used in section 715.

702. Include files.

```
⟨ Include files 10 ⟩ +≡
#include "localldf.h"
#include "nonwin.h"
#include "parser.hxx"
#include "dttmtype.h"
#include "scanner.h"
#include "dtsrctyp.h"
#include "querytyp.h"
#include "idtype.h"
#include "scnrtype.h"
```

703. Global variables. [LDF 2006.10.20.]

Log

[2006.10.20.] Added this section.

[LDF 2006.11.21.] Added **unsigned short** *tex_file_ctr*, **string** *tex_filename_str*, **ofstream** *tex_file_strm*, and **Mutex_Type** *tex_mutex*.

[LDF 2006.11.30.] Added **string** *copyright_tex_str*.

[LDF 2006.11.30.] Added **Mutex_Type** *time_mutex*.

```
⟨ Global variables 698 ⟩ +≡
  Mutex_Type cerr_mutex;
  Mutex_Type cout_mutex;
  Mutex_Type time_mutex;
  ofstream tex_file_strm;
  Mutex_Type tex_mutex;
```

```
unsigned short tex_file_ctr;
string tex_filename_str;
string copyright_tex_str;
```

704. Scanning and parsing input. [LDF Undated.]

705. *yyparse* declaration. [LDF Undated.]

⟨ Declare scanning and parsing functions 705 ⟩ ≡

```
int yyparse(void *);
```

See also sections 706 and 708.
This code is used in section 715.

706. *yywrap*.

⟨ Declare scanning and parsing functions 705 ⟩ +≡

```
#if 0
  int yywrap(void);
#endif
```

707.

⟨ Define scanning and parsing functions 707 ⟩ ≡

```
#if 0
  int yywrap(void)
  {
    return 1;
  }
#endif
```

See also section 709.
This code is used in section 715.

708. *yyerror*. [LDF 2006.10.20.]

⟨ Declare scanning and parsing functions 705 ⟩ +≡

```
void yyerror(const char *s);
```

709.

⟨ Define scanning and parsing functions 707 ⟩ +≡

```
void yyerror(const char *s)
{
  return;
}
```

710. Main itself. [LDF 2006.10.20.]

Log

[LDF 2006.10.20.] Added this function.

[LDF 2006.11.02.] No longer closing *scanner_node→in_strm* and *scanner_node→log_strm*. This is now done in the **Scanner_Node** destructor.

[LDF 2006.11.02.] Now calling *scanner_node→initialize_id_map*.

[LDF 2006.11.21.] Now initializing the global variable **unsigned short** *tex_file_ctr* to 0.

⟨ Define *main* 710 ⟩ ≡

```
  int main(int argc, char *argv[]){
#if 0     /* 1 */
      unsigned long long temp_val = 4096LL;
      bitset < 64 > b;
      b.flip();
      cerr << "`b'␣==␣" << b << endl << "`temp_val'␣==␣" << temp_val << "b␣&␣temp_val␣==␣" <<
          (b & temp_val) << endl;
      getchar();
      cerr << "`sizeof(unsigned␣long␣long)'␣==␣" << sizeof(unsigned long
          long) << endl << "`ULONG_LONG_MAX'␣==␣" << ULONG_LONG_MAX << endl;
      getchar();
      for (int i = 27; i < 36; ++i) cerr << (long long) pow((double) 2, i) << "\n";
      getchar();
#endif
#if 0     /* 1 */
#define DEBUG_OUTPUT
#else
#undef DEBUG_OUTPUT
#endif
      copyright_tex_str = "%%␣*␣(1)␣Copyright␣and␣License.\n\n";
      copyright_tex_str += "%%␣This␣file␣is␣part␣of␣the␣IWF␣Metadata␣Harvester,␣";
      copyright_tex_str += "a␣package␣for␣metadata␣harvesting.\n";
      copyright_tex_str += "%%␣Copyright␣(C)␣2006,␣2007␣IWF␣Wissen␣und␣Medien␣";
      copyright_tex_str += "gGmbH\n\n";
      copyright_tex_str += "%%␣The␣auth\pi or␣is␣Laurence␣D.␣Finston.\n\n";
      copyright_tex_str += "%%␣The␣IWF␣Metadata␣Harvester␣is␣free␣software;␣";
      copyright_tex_str += "you␣can␣redistribute␣it␣and/or␣modify\n";
      copyright_tex_str += "%%␣it␣under␣the␣terms␣of␣the␣GNU␣General␣Public␣License␣";
      copyright_tex_str += "as␣published␣by\n";
      copyright_tex_str += "%%␣the␣Free␣Software␣Foundation;␣either␣version␣2␣of␣the␣";
      copyright_tex_str += "License,␣or\n";
      copyright_tex_str += "%%␣(at␣your␣option)␣any␣later␣version.\n\n";
```

```
copyright_tex_str += "%%␣The␣IWF␣Metadata␣Harvester␣is␣distributed␣in␣";
copyright_tex_str += "the␣hope␣that␣it␣will␣be␣useful,\n";
copyright_tex_str += "%%␣but␣WITHOUT␣ANY␣WARRANTY;␣without␣even␣the␣implied␣";
copyright_tex_str += "warranty␣of\n";
copyright_tex_str += "%%␣MERCHANTABILITY␣or␣FITNESS␣FOR␣A␣PARTICULAR␣PURPOSE.␣␣";
copyright_tex_str += "See␣the\n";
copyright_tex_str += "%%␣GNU␣General␣Public␣License␣for␣more␣details.\n\n";
copyright_tex_str += "%%␣You␣should␣have␣received␣a␣copy␣of␣the␣GNU␣General␣";
copyright_tex_str += "Public␣License\n";
copyright_tex_str += "%%␣along␣with␣the␣IWF␣Metadata␣Harvester;␣if␣not,␣";
copyright_tex_str += "write␣to␣the␣Free␣Software\n";
copyright_tex_str += "%%␣Foundation,␣Inc.,␣51␣Franklin␣St,␣Fifth␣Floor,␣Boston,␣MA␣␣";
copyright_tex_str += "02110-1301␣␣USA\n\n";
copyright_tex_str += "%%␣The␣IWF␣Metadata␣Harvester␣is␣available␣for␣downloading␣\
    from␣the\n";
copyright_tex_str += "%%␣following␣FTP␣server:\n";
copyright_tex_str += "%%␣ftp://ftp.gwdg.de/pub/gnu2/iwfmdh/\n\n";
copyright_tex_str += "%%␣Please␣send␣bug␣reports␣to␣lfinsto1@gwdg.de\n\n";
copyright_tex_str += "%%␣The␣author␣can␣be␣contacted␣at:\n\n";
copyright_tex_str += "%%␣Laurence␣D.␣Finston\n";
copyright_tex_str += "%%␣Kreuzbergring␣41\n";
copyright_tex_str += "%%␣D-37075␣Goettingen\n";
copyright_tex_str += "%%␣Germany\n\n";
copyright_tex_str += "%%␣lfinsto1@gwdg.de\n";
copyright_tex_str += "%%␣s246794@stud.uni-goettingen.de\n";
copyright_tex_str += "\n\n";

using namespace Scan_Parse;
stringstream temp_strm;
int status;

status = initialize_maps();
status = Query_Type::initialize_type_maps();
if ((status = Query_Type::initialize_flags()) ≠ 0) {
  cerr ≪ "ERROR!␣In␣'main':" ≪ endl ≪ "'Query_Type::initialize_flags'␣returned\
      ␣unsuccessfully,␣" ≪ "with␣return␣value␣" ≪ status ≪ endl ≪
      "Exiting␣'main'␣unsuccessfully␣with␣return␣value␣1." ≪ endl;
  return 1;
}
status = Datasource_Type::initialize_type_maps();
tex_file_ctr = 0;
tex_filename_str = "scttex_";

Scanner_Type *scanner_node = new Scanner_Type;

status = Id_Type::initialize_subtype_map(scanner_node);
```

See also sections 711, 712, and 713.

This code is used in section 715.

711. Process command line options. [LDF 2006.11.03.]

Options:

b — Enable GNU Bison's tracing facilities.

Log

[2006.11.03.] Added this section.

```
⟨ Define main 710 ⟩ +≡
  for (int curr_opt = 0; curr_opt >= 0; ) {
    curr_opt = getopt(argc, argv, "b");
#ifdef DEBUG_OUTPUT
    temp_strm << "`curr_opt'  == " << curr_opt << endl;
    cerr_mutex.Lock();
    cerr << temp_strm.str();
    cerr_mutex.Unlock();
    scanner_node->log_strm << temp_strm.str();
    temp_strm.str("");
#endif
    switch (curr_opt) {
    case 'b': yydebug = 1;
      break;
    default: break;
    }   /* switch */
  }   /* for */
```

712. Process filename argument. [LDF 2006.11.03.]

Log

[2006.11.03.] Added this section.

```
⟨ Define main 710 ⟩ +≡
#if 0   /* 1 */
#ifdef DEBUG_OUTPUT
  temp_strm ≪ "‘argc’␣==␣" ≪ argc ≪ ",␣‘optind’␣==␣" ≪ optind ≪ endl;
  if (argv[optind] ≡ 0) temp_strm ≪ "No␣filename␣argument.";
  else temp_strm ≪ "Filename␣argument␣==␣‘" ≪ argv[optind] ≪ "’";
  cerr_mutex.Lock( );
  cerr ≪ temp_strm.str( );
  cerr_mutex.Unlock( );
  scanner_node→log_strm ≪ temp_strm.str( );
  temp_strm.str("");
  getchar( );
#endif
#endif
  if (argc > optind) {
    strcpy(scanner_node→in_filename, argv[optind]);
  }
  else strcpy(scanner_node→in_filename, "commands.txt");
  scanner_node→in_strm.open(scanner_node→in_filename);
  strcpy(scanner_node→log_filename, "log_1.txt");
  scanner_node→log_strm.open(scanner_node→log_filename);
  scanner_node→log_strm ≪ "%%␣log_1.txt" ≪ endl ≪ endl ≪ copyright_tex_str ≪ endl;
```

713.

```
⟨ Define main 710 ⟩ +≡
  scanner_node→initialize_id_map( );
#ifdef DEBUG_OUTPUT
  status = show_keyword_map(scanner_node);
#endif
  status = yyparse(scanner_node);
  delete scanner_node;
  scanner_node = 0;
  return 0;
#undef DEBUG_OUTPUT
  }   /* End of main definition. */
```

714. Putting Main together.

715. This is what's compiled.

⟨ Include files 10 ⟩
⟨ `main.web` 701 ⟩
using namespace std;
⟨ Global variables 698 ⟩
⟨ Declare scanning and parsing functions 705 ⟩
⟨ Define *main* 710 ⟩
⟨ Define scanning and parsing functions 707 ⟩

716. GNU General Public License. [LDF 2006.10.23.]

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> ⟨one line to give the program's name and a brief idea of what it does.⟩
> Copyright (C) ⟨year⟩ ⟨name of author⟩
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright © year name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

⟨signature of Ty Coon⟩, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

⟨ GNU General Public License 716 ⟩ ≡ /* This section contains no C++ code. */

This code is cited in section 2.

This code is used in section 718.

717. GNU Free Documentation License. [LDF 2006.10.23.]

GNU Free Documentation License
Version 1.2, November 2002

Copyright © 2000,2001,2002 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

⟨GNU Free Documentation License 717⟩ ≡ /* This section contains no C++ code. */

This code is cited in section 2.

This code is used in section 718.

718. Include sections without C++ code. These sections contain no code, or only commented-out code. However, they must be included somewhere, or CWEAVE will issue warnings. [LDF 2006.09.27.]

⟨ GNU General Public License 716 ⟩
⟨ GNU Free Documentation License 717 ⟩

719. Index.

IWF Metadata Harvester III
Scantest: The Program
Version 1.0

by Laurence D. Finston

December 2006