The NCBI C++ Software Development Toolkit

Michel Dumontier

Blueprint Initiative

mjdumont@blueprint.org

A Powerful Bioinformatics SDK

- Created and Maintained by the National Center for Biotechnology Information (NCBI) -Information Engineering Branch (IEB)
- Provides an extensive set of functions to create, store and compute biological information
- Well designed, well documented.
- Cross platform, open-source & GNU GPL
- C++ code generator to convert ASN.1/XML specification and data
- Object manager to deal with sequence manipulation, annotation and retrieval.
- Powerful stand-alone, client-server and CGI application framework
- Easy to integrate individual projects with existing NCBI code
- Plenty for Everyone!

Thanks!

NCBI

- **Jim Ostell**
  - **Intellectual Grandfather**
- **Denis Vakatov – Manager**
- **Full Time Developers**
  - Eugene Vasilchenkov
  - Anton Lavrentiev
  - Aleksey Grichenko
  - Aaron Ucko
  - Andrei Gourianov
  - Vladimir Ivanov
- **Major Contributors**
  - Anton Butanaev
  - Mike DiCuccio
  - Joanathan Kans
  - Michael Kholodov
  - Mikhael Kimelman
  - Vladimir Soussov
  - Paul Thiessen
- and many more…

Blueprint Initiative

- Chris Hogue
  - PI & Intellectual Father
- Research
  - Doron Betel (C++)
- BIND Group
  - Marc Dumontier
- SeqHound Group
  - Katerina Michalikova
- Structure Group
  - Howard Feldman
- Visualization Group
  - Brigitte Tuekam
- and many more…

Outline

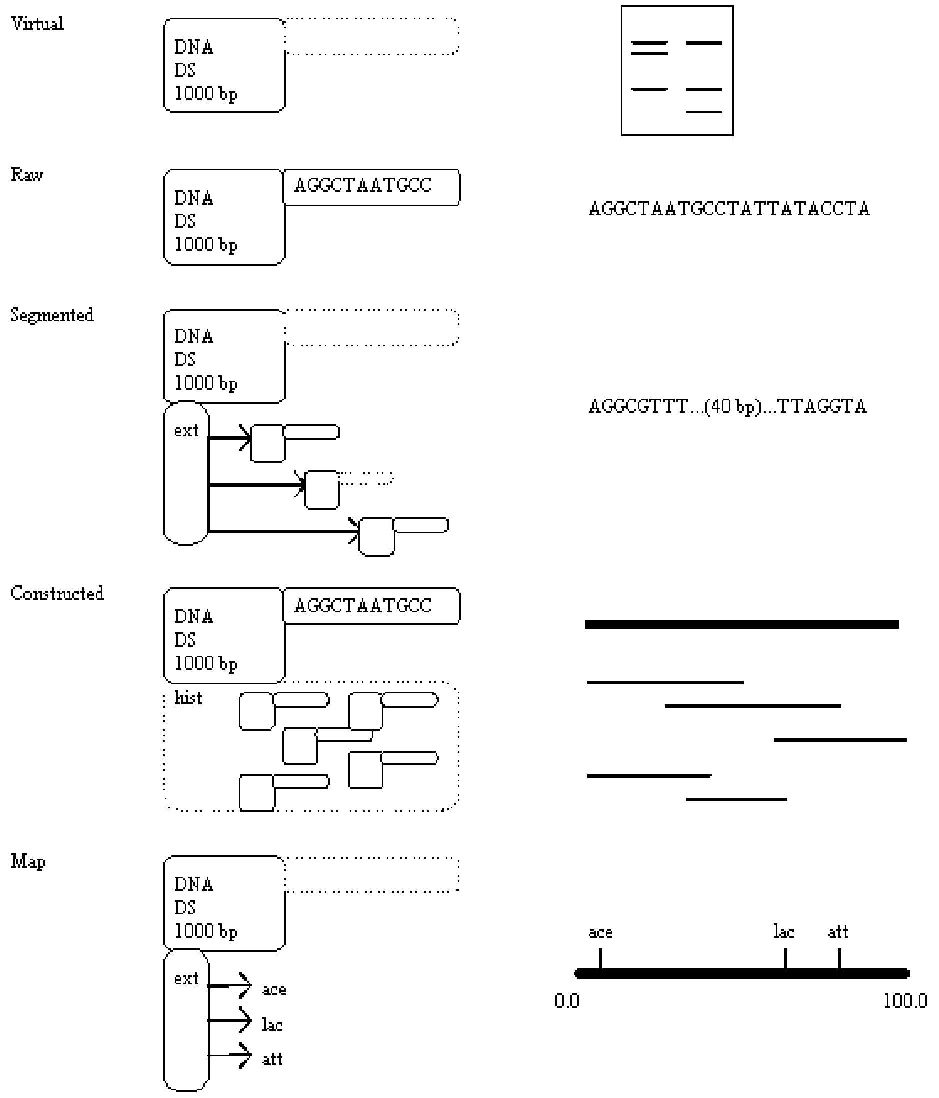
- The NCBI Data Model
- The Object Manager
- The Application Framework
- CGI Development
- The Database API
- The Modules
- Resources

The NCBI Data Model

- Molecular biology spans most fields of biology from evolution to development, from enzymology to agriculture, from statistical mechanics to medicine.

- The NCBI data model establishes a biological sequence as a simple integer coordinate system with which diverse data can be associated. It is reasonable to hope that such a simple core can be very stable and compatible with a very wide range of data.

NCBI Data Model
Instance of a Sequence



The NCBI Data Model
The Biological Sequence

Biological sequence data and its associated information are described in NCBI data model using Abstract Syntax Notation One (ASN.1).

The Bioseq is composed of

- list of identifiers
  - GI, accession
- optional descriptors
  - taxonomy, publications
- Sequence instantiation
  - Type (dna, rna, protein)
  - Topology (ss,ds)
  - Virtual, segmented
- Annotation
  - Gene, mRNA, CDS
  - SNP,STS

<u>ASN.1 Specification</u>

```
Bioseq ::= SEQUENCE {
    id SET OF Seq-id ,
    descr Seq-descr OPTIONAL ,
    inst Seq-inst ,
    annot SET OF Seq-annot
        OPTIONAL
}
```

http://www.ncbi.nlm.nih.gov/IEB/ToolBox/SDKDOCS/DATAMODL.HTML

ASN.1 record for GI 98

```
bioseq {
  id {
    embl {
      name "BTATP2" ,
      accession "X05219" ,
      version 1 } ,
    gi 98 } ,
  descr {
    title "B.taurus mRNA for mit-ATP synthase
           proteolipid P2 subunit precursor" ,
    embl {
      div mam ,
      keywords {
        "ATP synthase" } ,
        xref {
         {
          dbname
            name "GOA" ,
          id {
            str "P07926" } } ,
         {
          dbname
            code swissprot ,
          id {
            str "P07926" ,
            str "AT92_BOVIN" } } } } ,
     molinfo {
       biomol mRNA } } ,
```

```
inst {
  repr raw ,
  mol rna ,
  length 615 ,
  seq-data
    ncbi2na
     'E5B257D355E00EC47E650BDB75155D7E34A205DC4B3E2
     58D1EDE4BAEB086168047868E2251249FA4B2F55B55E14
     5D1F1D725927D4052E53F42A13E1124942F4FA27AA7944
     B2ABA7A77A27A0FA85BBFA8B74D3EBCE52815F77824927
     7DD7194F7A9FE5776853AA77FE5E3AE97F74D77D94EE0A
     25BF51754CBDF756EDD3795ECEFDFF5EC5D54A417AA02C
     BE9D2A7E120A0840C0C7B3C308000400'H } ,
 annot {
    {
     data
       ftable {
         {
          data
            imp {
              key "misc_feature" } ,
            comment "put.polyA signal" ,
            location
              int {
                from 582 ,
                to 587 ,
                id
                  gi 98 } } } } } } ,
```

ASN.1 ⇔ XML Schema

The NCBI's *datatool* program generates corresponding data specifications, c++ code handlers and data conversions.

ASN.1 Specification

```
Bioseq ::= SEQUENCE {
        id SET OF Seq-id ,
        descr Seq-descr
             OPTIONAL ,
        inst Seq-inst ,
        annot SET OF Seq-annot
             OPTIONAL
}
```

XML Schema

```
<xs:element name="Bioseq">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Bioseq_id"/>
      <xs:element ref="Bioseq_descr" minOccurs="0"/>
      <xs:element ref="Bioseq_inst"/>
      <xs:element ref="Bioseq_annot" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Converting ASN.1 and XML

```
// copy a Bioseq in ASN.1 text to XML
auto_ptr<CObjectIStream>
  txt_in(CObjectIStream::Open("my_seq.asn",eSerial_AsnText));

auto_ptr<CObjectOStream>
  xml_out(CObjectOStream::Open("my_seq.xml",eSerial_Xml));

CObjectStreamCopier xml_copier(*txt_in, *xml_out);
xml_copier.copy(CBioseq::GetTypeInfo());

// copy a Bioseq from XML to ASN.1 binary
auto_ptr<CObjectIStream>
  xml_in(CObjectIStream::Open("my_seq.xml",eSerial_Xml));
auto_ptr<CObjectOStream>
  bin_out(CObjectOStream::Open("my_seq.asb",eSerial_AsnBinary));

CObjectStreamCopier bin_copier(*xml_in, *bin_out);
bin_copier.copy(CBioseq::GetTypeInfo());
```

Outline

- The NCBI Data Model
- The Object Manager
- The Application Framework
- CGI Development
- The Database API
- The Modules
- Resources

The Object Manager

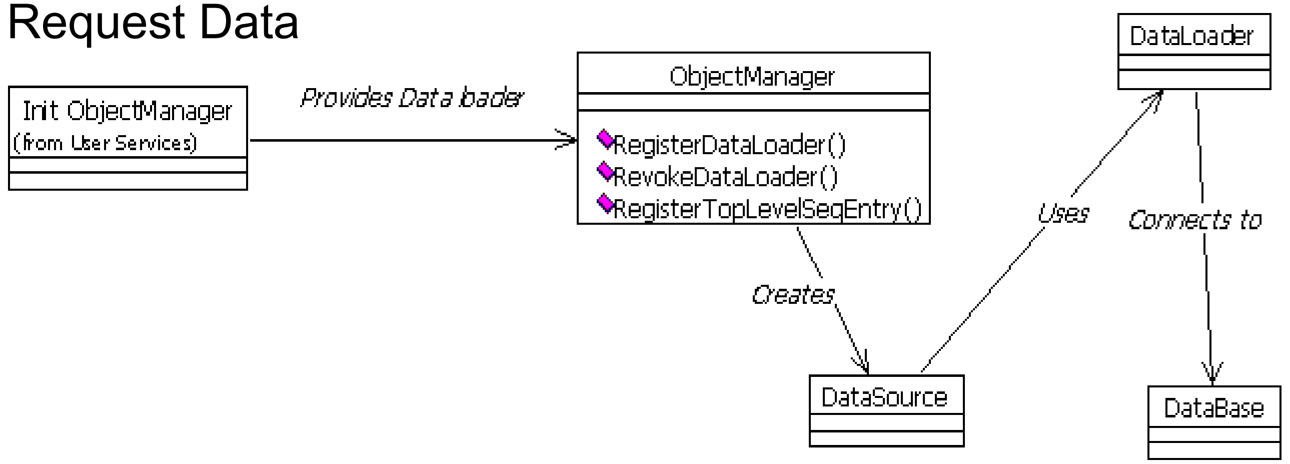
- The Object Manager module is a library of C++ classes, which facilitate access to biological sequence data.

- The Object Manager has been designed to present an interface to users that minimizes their exposure to the details of interacting with biological databases and their underlying data structures.

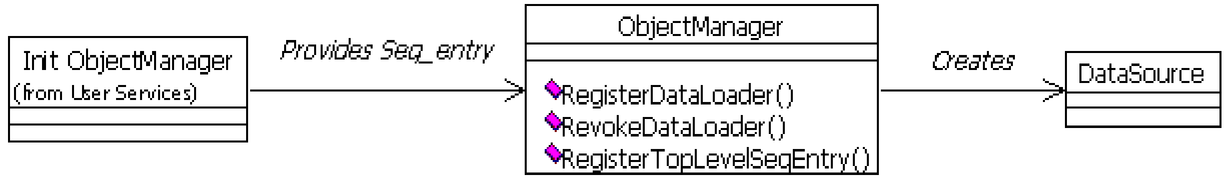
- It makes it possible to transparently download data from the GenBank database, investigate biological sequence data structure, retrieve sequence data, descriptions and annotations.

Object Manager Initialization

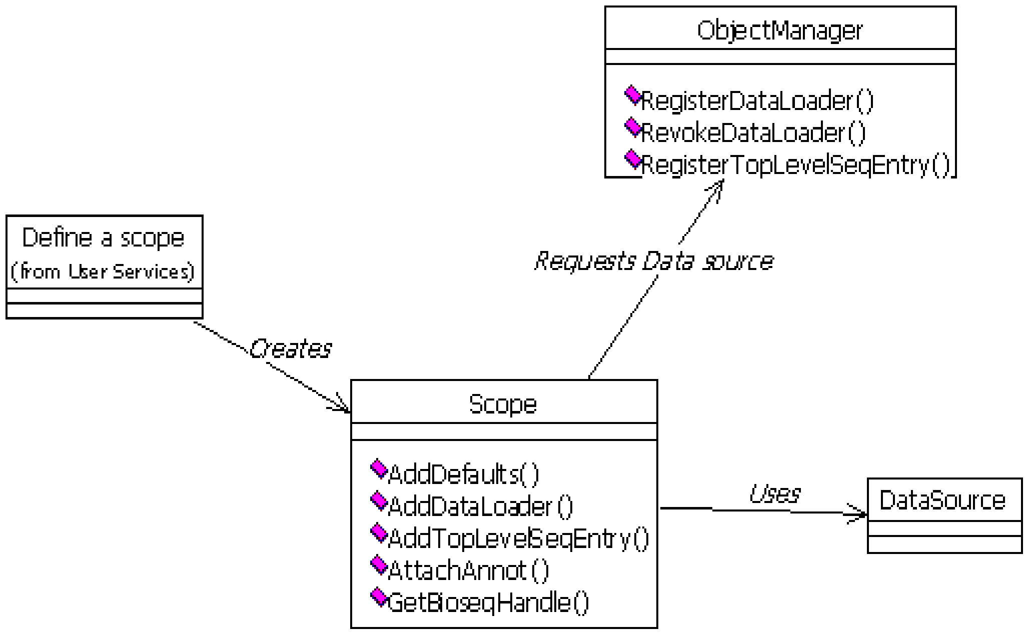
CASE 1: Request Data



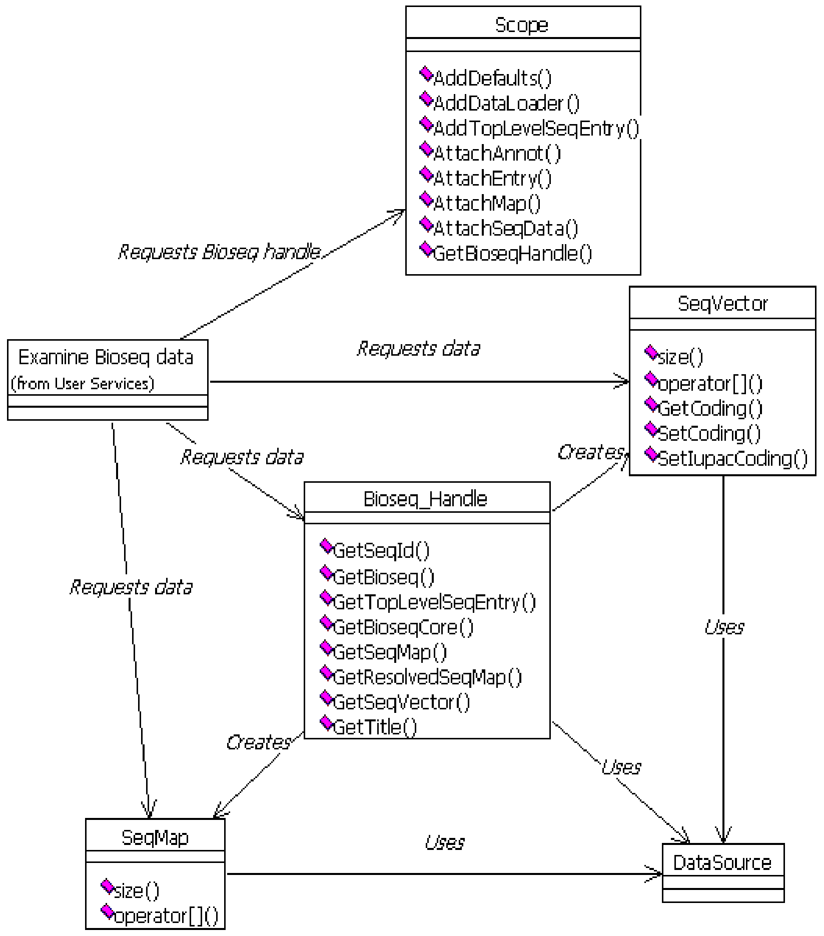
CASE 2: Use Local Data



Defining A Data View
The Scope



Getting a Handle on the Sequence *Bioseq_Handle*



Using the Object Manager

```
#include <objects/objmgr/object_manager.hpp>
#include <objects/objmgr/scope.hpp>
#include <objects/objmgr/bioseq_handle.hpp>
#include <objects/objmgr/seq_vector.hpp>
#include <objects/objmgr/desc_ci.hpp>
#include <objects/objmgr/feat_ci.hpp>
#include <objects/objmgr/align_ci.hpp>
#include <objects/objmgr/gbloader.hpp>
#include <objects/objmgr/reader_id1.hpp>
```

```
// instantiate the object manager
CRef<CObjectManager> obj_mgr = new CObjectManager;
// add a data loader
obj_mgr->RegisterDataLoader(*(new CGBDataLoader("ID")),
   CObjectManager::eDefault);
// define a scope and add the NCBI data loader
CRef<CScope> scope = new CScope(*obj_mgr);
scope.AddDataLoader("GENBANK");
```

Using the Object Manager

```
// now get the sequence for GI 98
int gi = 98;
CSeq_id seqid;
seqid.SetGi(gi);
CBioseq_Handle handle = scope.GetBioseqHandle(seqid);
CSeqVector seq_vec = handle.GetSeqVector();
string sequence;
for (size_t i = 0; i < seq_vec.size(); i++) {
   sequence += seq_vec[i];
}
NcbiCout << "GI " << gi << " sequence:\n" << sequence <<
   "\n";
```

Sequence Map

The Sequence map is a collection of segments, which describe sequence parts in general - location and type only, without providing any real data.

It is possible then to enumerate all the segments in the map asking their type, length or position:

```
const CSeqMap& seqmap = handle.GetSeqMap();
int len = 0;
for (size_t i = 0; i < seq_map.size(); i++) {
    switch (seq_map[i].GetType()) {
        case CSeqMap::eSeqData:
            len += seq_map[i].GetLength();
            break;
        case CSeqMap::eSeqRef:
            len += seq_map[i].GetLength();
            break;
        case CSeqMap::eSeqGap:
            len += seq_map[i].GetLength();
            break;
        default:
            break;
}}
```

The NCBI C++ Iterators

- Similar to the STL iterators, the NCBI C++ TK implemented type specific iterators that can be used to traverse ASN.1 object.

- This mechanism allows for easy access of specific data members nested within a given object. The iterator will extract all the data elements of a specific type that are nested several levels deep.

- Iteration can also be over a set of data members. For example, one can collect all the sequence id’s and feature tables for a number of sequences in one iteration.

```
BIND-Pathway::={ pub {…},
          interactions{
                  { iid, 118, pub { ..},
                    a,  {egf,    pub { ..}},
                    b, {egfr,    pub { ..}},
                    descr{
                              { place
                                    start membrane,
                                    end cytoplasm
                              }
                    }
                  }
                  { iid, 220 , pub { ..},
                    a, {egf-egfr,  pub { ..}},
                    b, {ATP,       pub { ..}},
                    descr{
                              { place
                                    start cytoplasm,
                                    end  cytoplasm
                              }
                    }
                  }
                  { iid, 238, pub { ..},
                    a, {GRB2,    pub { ..}},
                    b, {SOS1,    pub { ..}},
                    descr{
                              { place
                                    start cytoplasm,
                                    end  nucleus
                    }}}}
```

Traversing through objects

```
CBIND_pathway pathway;

CTypeIterator<CPub> pub_i;

for(pub_i =Begin(pathway); pub_i; ++ pub_i)
{
  PrintAuthorList(* pub_i );
}
```

Outline

- The NCBI Data Model
- The Object Manager
- The Application Framework
- CGI Development
- The DBAPI
- The Modules
- Resources

NCBI application classes

Five fundamental classes form the foundation of an NCBI C++ toolkit application.

1. *CNcbiApplication* – the Big Kahuna!
   - Provides the mechanism to execute the application.
   - Contains a data structure to get, hold and validate the program command-line arguments.
   - Contains a data structure to hold environment variables.
   - When and where errors are reported.
   - Contains methods to read, and modify config files.

The other four..

2. *CNcbiArguments* – holds the application's command-line arguments, along with methods for accessing and modifying them.

3. *CNcbiEnvironment* – store, access, and modify the environment variables accessed by the C library routine getenv().

4. *CNcbiRegistry* – load, access, modify and store runtime information read from a configuration file.

5. *CNcbiDiag* – class implements much of the functionality of the NCBI C Toolkit error processing mechanisms.

Using the Application Framework

```
#include <corelib/ncbiapp.hpp>

BEGIN_NCBI_SCOPE

class CTestApp : public CNcbiApplication {
 public:
   virtual int Run(void);
};

int CTestApp::Run()
 {
   NcbiCout << "Executing CTestApp::Run()!"
            << NcbiEndl;
   return 0;
 }
END_NCBI_SCOPE



USING_NCBI_SCOPE;
int main(int argc, const char* argv[])
 {
   CTestApp test;
   return test.AppMain(argc, argv);
 }
```

// within NCBI namespace

Our application implements the CNcbiApplication framework with the

mandatory implementation of virtual method Run

Use portable stream and macro typedefs

// end NCBI namespace

// use the NCBI namespace

Instantiate our class

Run the framework AppMain method – calls Run method

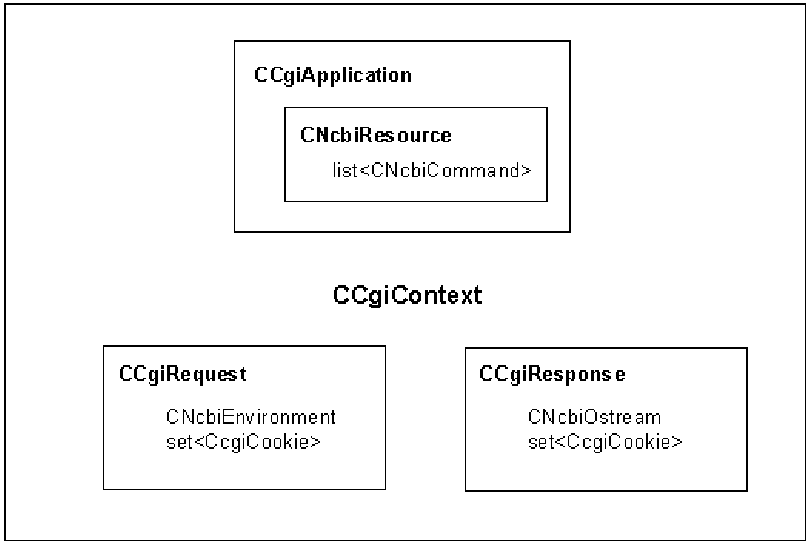
Outline

- The NCBI Data Model
- The Object Manager
- The Application Framework
- CGI Development
- The Database API
- The Modules
- Resources

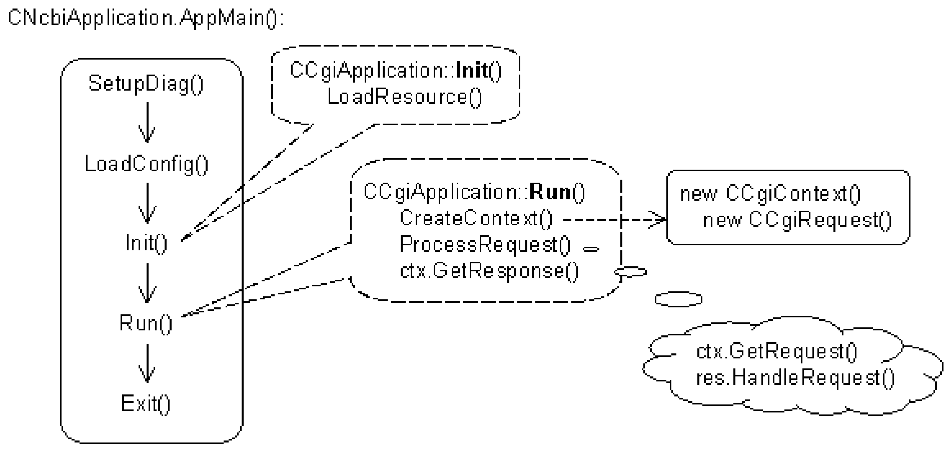
CGI

CGI Framework Features:

- A way to retrieve and store the current values of environment variables
- A means of retrieving and interpreting the client's query request string
- Mechanisms to service and respond to the requested query
- Methods and data structures to obtain, store, modify, and send cookies
- A way to set/reset the context of the application (for Fast-CGI)



The CCgiApplication class



- CCgiApplication derived from CNcbiApplication.
- Implements its own Run() function.
  - creates a context,
  - processes the request
  - calls the context to send the response

Programmer’s responsibility

- Implement ProcessRequest method that uses the CCgiContext object to
  - Read the environment variables
  - Process the request
  - Create a response

CNcbiResource

CNcbiResource

- Handles the request. Created by registry object which defines data paths, resources and env. variables.
- Holds list of CNcbiCommand commands. HandleRequest(ctx) looks at the command list and executes the first match.

CCgiRequest

- Request class acts as an interface between the user’s query and the CGI program.
- Caches environment, server, client and request information.
- Knows about POST and GET and will process the request accordingly.

CCgiResponse

- Provides an interface to the program output stream.
- Generates the appropriate HTML header (MIME-type).
- All cookies that are to be sent to the client are included in the header output.

Simple CGI

```
#include <ncbi_pch.hpp>
#include <cgi/ncbicgi.hpp>
#include <cgi/ncbicgir.hpp>

// turn on the ncbi namespace
USING_NCBI_SCOPE;

class CMyCgi : public CCgiApplication
{
public:
    virtual int ProcessRequest(CCgiContext& ctx);
};


int main(int argc, char* argv[]) {
        return CMyCgi ().AppMain(argc, argv);
}

Int CMyCgi::ProcessRequest(CCgiContext& ctx)
{
  // Get the context request and response
  const CCgiRequest& Request  = ctx.GetRequest();
  CCgiResponse&      Response = ctx.GetResponse();

  // Get the list of CGI request name/value pairs
  const TCgiEntries& Entries = Request.GetEntries();

  // this program expects queries of the form
  mycgi?key=value
  string key = “key”;
  string value;
  TCgiEntries::const_iterator ciKey =
  Entries.find(key);
  if (ciKey == Entries.end()) value = “no value
  specified";
  else value = ciKey->second;

  // print out the results
  Response.out() << "<html><body>”;
  Response.out() << “Key: " << key << “<br/>”;
  Response.out() << “Value: " << value;
  Response.out() << "</body></html>" << endl;
  Response.Flush()
  return 0;
}
```

Outline

- The NCBI Data Model
- The Object Manager
- The Application Framework
- CGI Development
- The Database API
- The Modules
- Resources

Database API

- The NCBI DBAPI driver library describes and implements a set of objects needed to provide a uniform low-level access to the various relational database management systems (RDBMS).

- The basic driver functionality is the same as in most other RDBMS client APIs.
  - Open a connection to a server
  - Execute a command (query) on this connection
  - Fetch results.

- The main advantage of using the driver is that you don't have to change your own upper-level code if you need to move from one RDBMS client API to another.

DBAPI Driver Components

- The Driver Manager
  - Allows a mixture of statically linked and dynamically loaded drivers and use them together in one executable.

- Driver Context
  - Effectively a RDBMS dependent "Connection" factory
  - All driver contexts implement the same interface defined in *I_DriverContext* class
  - Connection is put into RDBMS independent object *CDB_Connection*

- Error Handling
  - The NCBI DBAPI driver intercepts all those error messages in all different formats and converts them into the objects of CDB_Exception derived types

Supported DBAPI drivers

- Sybase CTLIB
- Sybase DBLIB
- Microsoft DBLIB
- FreeTDS 0.60 (TDS ver. 8.0)
- ODBC
- MySQL Driver

DBAPI User Components

- Driver Manager
  - Register available drivers with Driver Manager
- Data Source and Connections
  - The *IDataSource* interface defines the database platform.
  - An IDataSource can create objects represented by an *IConnection* interface, which is responsible for the connection to the database.

```
IDataSource *ds = dm.CreateDs(“odbc”);
IConnection *conn = ds->CreateConnection();
conn->Connect("user", "password", "server","database");
IStatement *stmt = conn->CreateStatement();
stmt->Execute("select * FROM my_table");
```

Handling Results

- CVariant
  - Represents any database data type (except BLOBs).
  - It is an object, not a pointer, so it behaves like a primitive C++ type.
    - Basic comparison operators are supported (==, !=, <) for identical internal types.
    - If types are not identical, CVariantException is thrown.
    - CVariant has a set of getters to extract a value of a particular type, e.g. GetInt4(), GetByte(), GetString(), etc. If GetString() is called for a different type, like DateTime or integer it tries to convert it to a string. It it doesn't succeed, CVariantException is thrown.

Using the DBAPI

```
stmt->Execute("select id FROM my_table");
while( stmt->HasMoreResults() ) {
    // Get a Row result set
    if( stmt->HasRows() ) {
         IResultSet *rs = stmt->GetResultset();

         // Retrieve row results, if any
         while( rs->Next() ) {
             int col1 = rs->GetVariant(1).GetInt4();
             ...
         }
     }
}
```

DBAPI also supports …

- Stored Procedures
- Cursors
- BLOBS
- Bulk Insert Operations

Outline

- The NCBI Data Model
- The Object Manager
- The Application Framework
- CGI Development
- The Database API
- The Modules
- Resources

The NCBI C++ Modules

ALGORITHM - Needleman-Wunsch, cDNA/mRNA-to-genomic algorithm and BLAST C++ API.

CORELIB - Platform code, application frame work, argument processing, diagnostics and exception, templates utilities, threads, Object and Ref classes and much more.

CONNECT- A C++ interface to the standard sockets SOCK API.

CTOOLS- Bridging between the current C++ and old C toolkits.

More Modules

CGI- Defines a CGI-specific application, deals with incoming CGI requests, posting responses, CGI-context (for FAST CGI), and cookies read/write.

DBAPI- Provides a common interface to different RDBMS. Models a database as a data source that can be accessed and queried through SQL. Supports Sybase (CTLIB/DBLIB), Microsoft DBLIB, FreeTDS and ODBC.

GUI- Basic functionality to display, navigate and highlight sequences. Uses OpenGL and FLTK third party graphic libs.

More Modules…

HTML- Generating HTML pages from a program, HTML tag classes, interfacing with CGI applications, support for using template HTML pages.

OBJECT MANAGER- facilitate access to biological sequence data. It can transparently download data from the GenBank database, investigate biological sequence data structure, retrieve sequence data, descriptions and annotations.

Even more modules!

SERIAL- ASN.1 serialization. Deals with all aspects for reading, writing, and transferring between independent processes.

UTIL- Useful miscellaneous classes such as Checksum, Console debug dump, Lightweight string, Random number generator, string matching and more.

Outline

- The NCBI Data Model
- The Object Manager
- The Application Framework
- CGI Development
- The Database API
- The Modules
- Resources

Getting the Open-Source Toolkit

CVS

- Production Version and Release Notes
  - ftp://ftp.ncbi.nih.gov/toolbox/ncbi_tools++/CURRENT/RELEASE_NOTES.html
- Development Version
  - cvs -d:pserver:anoncvs@anoncvs.ncbi.nlm.nih.gov:/vault login
  - cvs -d:pserver:anoncvs@anoncvs.ncbi.nlm.nih.gov:/vault co internal/c++

Building the Toolkit

- Build Configurations
  - MT, static/DLL, 64 bit
- Numerous configuration possibilities
  - MySQL, ODBC, BDB, graphics, GUI, etc
- Unix
  - configure script
- Windows
  - MSVC++ 6.0 support is now deprecated
  - MSVC .NET

Resources

NCBI C++ SDK

- http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=toolkit

NCBI Data Model

- http://www.ncbi.nlm.nih.gov/IEB/ToolBox/SDKDOCS/DATAMODL.HTML

Blueprint NCBI C and C++ Lectures & Tutorials

- http://blueprint.org/products/toolkit/toolkit_course.html

Conclusion

- NCBI C++ SDK is a robust, powerful toolkit to build solid bioinformatics applications
- Future extensions
  - ASN.1 SOAP
  - More algorithms – sequence and structure
  - Genome Workbench