Babel 0.8.0 Release

Tammy Dahlgren, Tom Epperly, and Gary Kumfert

Center for Applied Scientific Computing

Common Component Architecture Working Group

January 16, 2003

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

UCRL-PRES-151471

Summary of new features & changes

- Initial F90 support
- SIDL backend
- Reentrant & unversioned packages
- New version syntax
- Usability improvements
- IOR additions
- Infrastructure changes

Initial Fortran 90 support

Recall that a minimalist approach was taken for quicker turn-around.

Feature	F77	F90	Comment
File extension	.f	.F90	Standard
Format	Fixed	Free	Although F90 handles both, the Impls are generated in free-form
Comment style	C	!	
Subroutine termination	end	end subroutine	
Use statement	---	New splicer block	
Subroutine name lengths	---	31 characters	Name mangling is employed

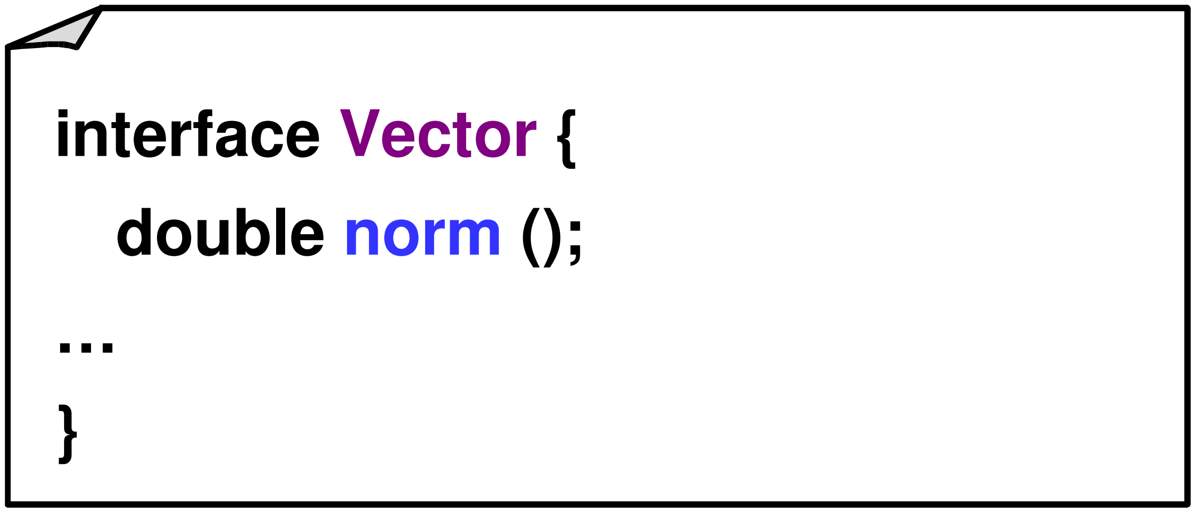
There have been a few changes since we last met.

- F90 binding changed to exploit use of *kind*
- Complete set of F90 regression tests (like F77’s)
- Build system modified
  - using “standard” autoconf macros for F90/F95
  - Automake 1.7.1 (includes macro name fix)
  - GNU m4-1.4q (includes overflow fix)

Modifying the build to support F90 required coordination with GNU tools developers to get necessary fixes.

- User’s Guide updated

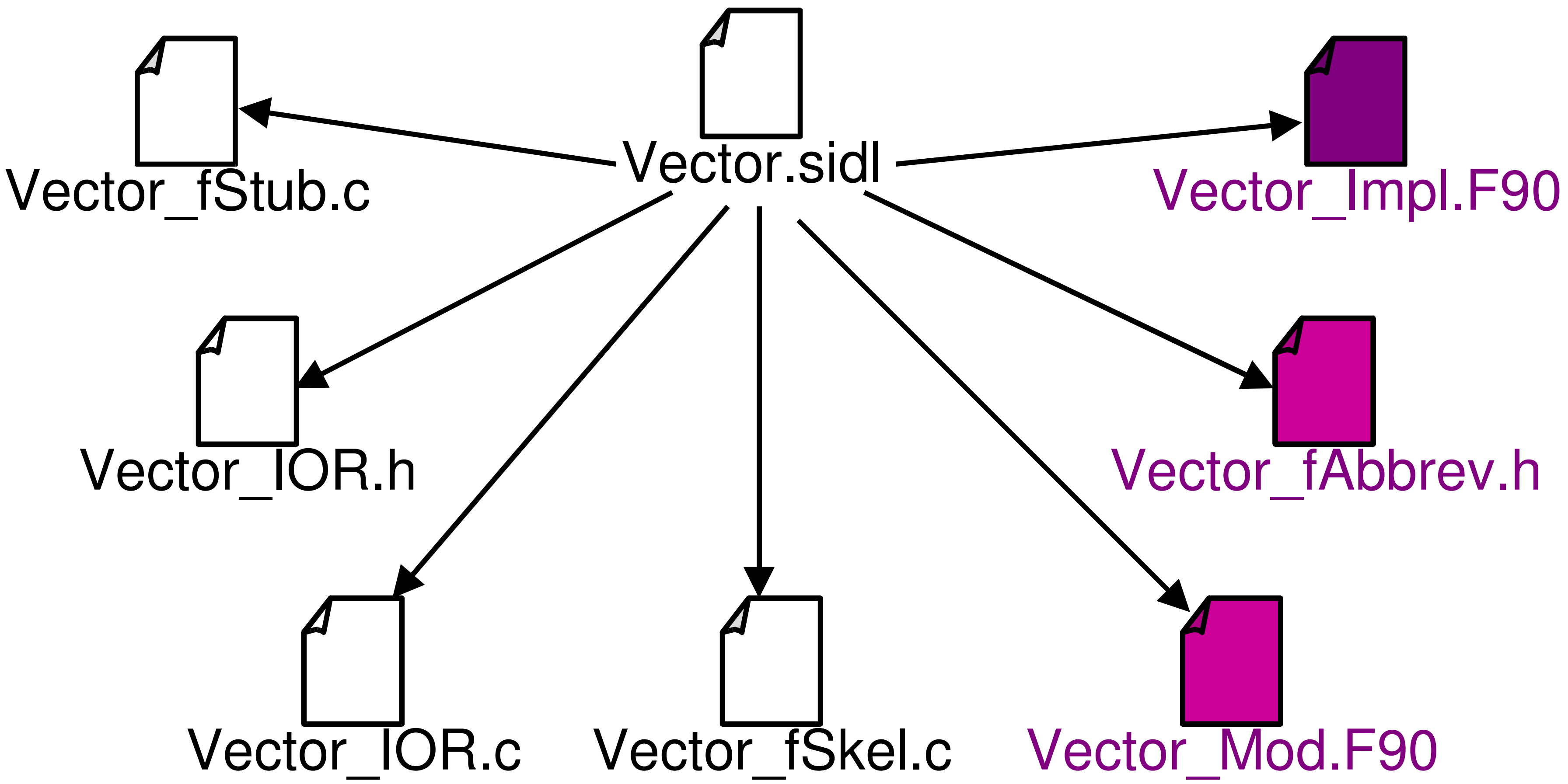
As an example, suppose we have a vector spec that includes a norm interface.



```
interface Vector {
   double norm ();
...
}
```

Vector.sidl

Generated F90 files still similar to their F77 counterparts but now have additional files.



CASC

The resulting Impl file snippet below illustrates the generated code.

```
#include Vector_fAbbrev.h
…
subroutine Vector_norm_mi(self, retval)
  ! DO-NOT-DELETE splicer.begin(Vector.norm.use)
  !    Insert use statements here…
  ! DO-NOT-DELETE splicer.end(Vector.norm.use)
  implicit none
  integer (selected_int_kind(18)) :: self
  real (selected_real_kind(15, 307)) :: retval

! DO-NOT-DELETE splicer.begin(Vector.norm)
!    Insert the implementation here…
! DO-NOT-DELETE splicer.end(Vector.norm)
end subroutine Vector_norm_mi
```

Vector_Impl.F90

The abbreviation header maps human readable method names to mangled ones.

```
#define Vector_somExcessivelyLongMethodName_m

  V_someExcessivejflax_vqhnrqww_m
#define vector_someexcessivelylongmethodname_m

  v_someexcessivejflax_vqhnrqww_m
#define VECTOR_SOMEEXCESSIVELYLONGMETHODNAME_M

  V_SOMEEXCESSIVEJFLAX_VQHNRQWW_M
```

Vector_fAbbrev.h

Finally, there’s a client-side module file snippet for the vector norm.

```
#include “Vector_fAbbrev.h”
…
module Vector
contains
  subroutine norm(self, retval)
  implicit none
  ! in Vector self
  integer (selected_int_kind(18)) :: self
  ! out double retval
  real (selected_real_kind(15, 307)) :: retval

  call Vector_norm_m(self, retval)
end subroutine norm
```

Vector_Mod.F90

Future Work

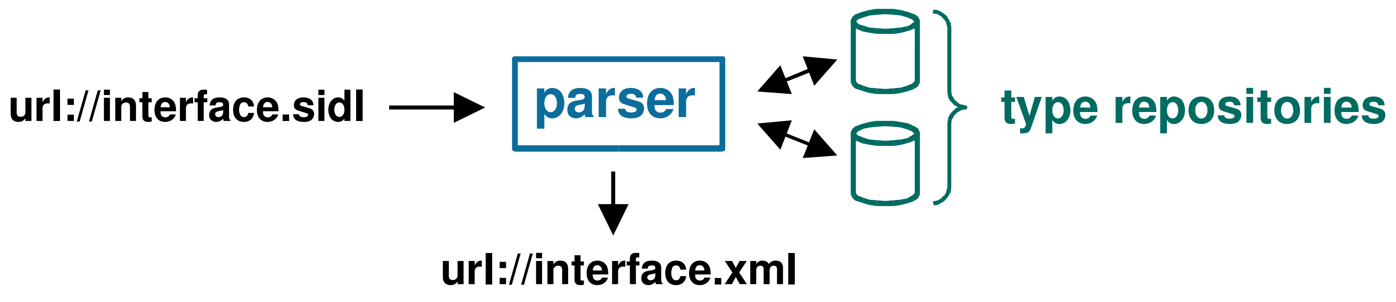
- Near term
  - Complete module files

- Long term
  - Address Fortran 90 array descriptors

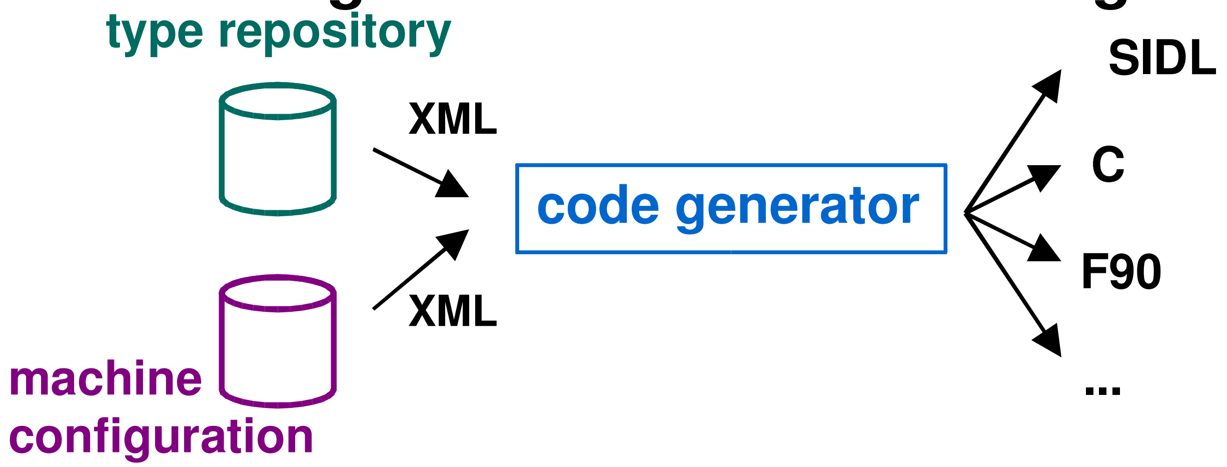
SIDL Backend

Babel can now generate SIDL files from compliant interface specifications.

- Recall Babel can be used to generate XML interfaces



- Now Babel can generate SIDL as well as glue code



Generated files do have some differences when compared to original SIDL files.

- One high-level package per file
  - *Even when* original had multiple such packages

- File name taken from high-level package name

  cca.sidl $\Rightarrow$ gov.sidl
  sidl.sidl $\Rightarrow$ SIDL.sidl

- implements-all becomes implements
  - Inherited methods are included instead

- Comments for enumeration values are lost

- White space differences include indentation, blank spaces and lines, and brace placement.

As an example, suppose we have a specification for package foo.

Original foo.sidl

```
package foo version 1.0 {

  class A { }

  package bar version 2.0 {
    class B { }
  }

}
```

Generated foo.sidl

```
package foo version 1.0 {

  class A {
  }

  package bar version 2.0 {

    class B {
    }

  }


}
```

To also illustrate the new version syntax, suppose we also have package fooTest.

Original fooTest.sidl

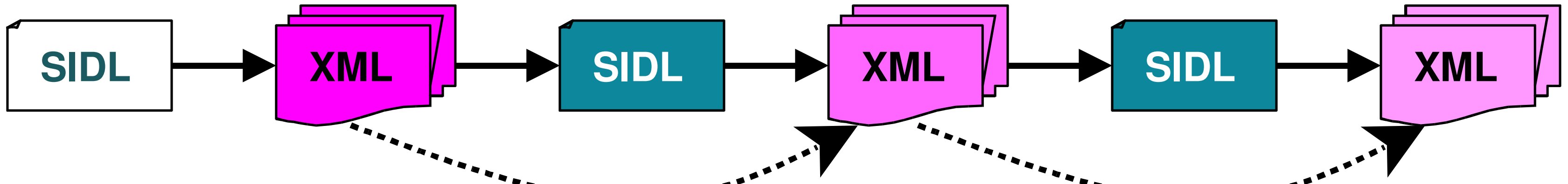
```
// An ignored comment
require foo version 1.0;
require foo.bar version 2.0;

/**
 * Test of comment with < & >.
 */
package fooTest version 0.1 {

  /**
   * An empty class.
   */
  class A extends foo.bar.B { }

  class B extends foo.A {}
}
```

Generated fooTest.sidl

```
require foo version 1.0;
require foo.bar version 2.0;

/**
 * Test of comment with < & >.
 */
package fooTest version 0.1 {

  /**
   * An empty class.
   */
  class A extends foo.bar.B
  {
  }

  class B extends foo.A
  {
  }

}
```

CASC

Tests of generated XML revealed only minor differences even after recursion.



- Metadata differences only
  - date — *unless* --suppress-timestamp used for both XML files
  - source-url
  - source-line — *unless* lines same in SIDL files used to generate the XML files

Continuing with the foo package example, the XML for foo is given below.

```xml
<?xml version=“1.0” encoding=“UTF-8”?>
<!DOCTYPE Symbol PUBLIC “-//CCA//SIDL Symbol DTD v1.1//EN” “SIDL.dtd”>
<Symbol>
  <SymbolName name=“foo” version=“1.0”/>
  <Metadata date=“20030110 10:58:21 PST”>
    <MetadataEntry key=“source-url” value=“file:/home/test/foo.sidl”/>
    <MetadataEntry key=“source-line” value=“1”/>
    <MetadataEntry key=“babel-version” value=“0.8.0”/>
  </Metadata>
  <Comment/>
  <Package final=“false”>
    <PackageSymbol name=“A” type=“class” version=“1.0”/>
    <PackageSymbol name=“bar” type=“package” version=“2.0”/>
  </Package>
</Symbol>
```

foo-v1.0.xml

And for class fooTest.A, which illustrates inheritance and comments.

```
<Symbol>
  <SymbolName name="fooTest.A" version="0.1"/>
  <Metadata date="20030110 10:58:41 PST">
    <MetadataEntry key="source-url" value="file:/home/test/fooTest.sidl"/>
    <MetadataEntry key="source-line" value="12"/>
    <MetadataEntry key="babel-version" value="0.8.0"/>
  </Metadata>
  <Comment>
An empty class.
  </Comment>
  <Class abstract="false">
    <Extends>
      <SymbolName name="foo.bar.B" version="2.0"/>
    </Extends>
    <ImplementsBlock/>
    <AllParentClasses>
      <SymbolName name="foo.bar.B" version="2.0"/>
      <SymbolName name="SIDL.BaseClass" version="0.8.0"/>
    </AllParentClasses>
    <AllParentInterfaces>
      <SymbolName name="SIDL.BaseInterface" version="0.8.0"/>
    </AllParentInterfaces>
  </Class>
</Symbol>
```

fooTest.A-v0.1.xml

The *--text* option has been added to enable generation of SIDL text.

Usage babel [-h | --help] or babel [-v | --version]
or babel *option(s) sidlfilename1* ... *sidlfilenameN*
where help, version, and option(s) are

-h	| --help	Display usage information and exit.
-v	| --version	Display version and exit.
-p	| --parse-check	Parse the sidl file but do not generate code.
-x	| --xml	Generate only SIDL XML (deprecated; use -tXML).
-c*lang*	| --client=*lang*	Generate only client code in specified language (C | C++ | F77 | F90 | Java | Python).
-s*lang*	| --server=*lang*	Generate server (and client) code in specified language (C | C++ | F77 | F90 | Python).
-t*form*	| --text=*form*	Generate only text in specified form (XML | SIDL), where XML updates the repository.
-o*dir*	| --output-directory=*dir*	Set Babel output directory ('.' default).
-R*path*	| --repository-path=*path*	Set semicolon-separated URL list used to resolve symbols.
-g	| --generate-subdirs	Generate code in subdirs matching package hierarchy.
--no-default-repository		Prohibit use of default to resolve symbols.
--suppress-timestamp		Suppress timestamps in generated files.
--generate-sidl-stdlib		Regenerate only the SIDL standard library.

Future Work

- Near term
  - Add new automated regression tests
  - Fill in new chapter in User’s Guide
- Long term
  - ***TBD***

Reentrant & unversioned packages

- Packages are now reentrant by default
- Packages can be declared as “final” to make them nonreentant
- Packages that only contain other packages can be unversioned

New version syntax

- In response to feedback from tutorial
- require x.y.z version 1.0;
- import x.y.z version 1.0;
  import x.y.z;
- package x version 1.0 {

  }

Usability improvements

- --vpath to indicate the source directory for the impl files
  - Separates hand written files from generated ones
- #line directives for easier debugging of C & C++ impl files

IOR & SIDL.BaseClass additions

- **SIDL.BaseClass stores IOR version for the class in its private data**
- **IOR now has function to retrieve IOR version**
- **SIDL.BaseClass has new getClassInfo() that returns**
- **SIDL.ClassInfo**

```
interface ClassInfo {
    /**
     * Return the name of the class.
     */
    string getName();

   /**
    * Get the version of the intermediate object representation.
    * This will be in the form of major_version.minor_version.
    */
    string getIORVersion();
 }
```

Infrastructure changes

- SIDL runtime library is separable
  - Separate configuration, compilation & distribution
- Babel testing using Gauntlet instead of Petf