



# libSBML-5

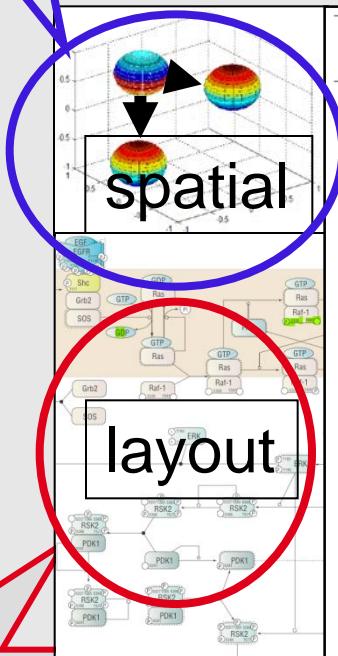
## How to write Level 3 packages extensions

Sarah Keating



possibly  
necessary

# SBML Level 3



additional  
information

## The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core

Michael Hucka (Chair) California Institute of Technology, US  
Frank Bergmann University of Washington, US  
Stefan Hoops Virginia Bioinformatics Institute, US  
Sarah M. Keating California Institute of Technology, US  
Sven Sahle University of Heidelberg, DE  
Darren J. Wilkinson Newcastle University, GB  
[sbml-editors@sbml.org](mailto:sbml-editors@sbml.org)  
**core**  
SBML Level 3 Version 1 Core

Release 1 (Candidate)

31 December 2009

Please report errors, ambiguities, and other issues in this document using the form at  
<http://sbml.org/specifications/sbml-level-3/version-1/core/issue-report-form>

Corrections and/or changes to this SBML language specification may appear over time.  
Notifications of new releases are broadcast on the mailing list [sbml-announce@sbml.org](mailto:sbml-announce@sbml.org)

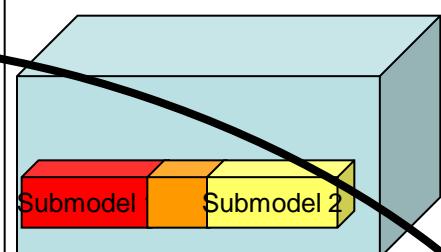
The latest release of the SBML Level 3 Version 1 Core specification is available at  
<http://sbml.org/specifications/sbml-level-3/version-1/core>

This release of the specification is available at  
<http://sbml.org/specifications/sbml-level-3/version-1/core/release-1/>

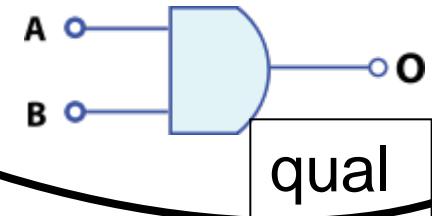
The list of known issues in all releases of SBML Level 3 Version 1 Core is available at  
<http://sbml.org/specifications/sbml-level-3/version-1/core/errata>

Formal schemas for use with the specification are available at  
<http://sbml.org/specifications/sbml-level-3/version-1/core/schemas>

mathematically necessary for  
correct interpretation



comp



# SBML Level 3

```
<sbml xmlns="http://www.sbml.org/sbml.level3/version1/core"  
      xmlns:comp="http://www.sbml.org/sbml/level3/version1/comp/version1"  
      comp:required="true"  
      xmlns:qual="http://www.sbml.org/sbml/level3/version1/qual/version1"  
      qual:required="true"  
      xmlns:layout="http://www.sbml.org/sbml/level3/version1/layout/version1"  
      layout:required="false"  
      xmlns:spatial="http://www.sbml.org/sbml/level3/version1/spatial/version1"  
      spatial:required="true"
```

# libSBML-5

libSBML core  
(very minor API changes from libSBML-4)

# libSBML-5

libSBML core  
(very minor API changes from libSBML-4)

```
typedef enum
{
    SBML_UNKNOWN

    , SBML_COMPARTMENT
    , SBML_COMPARTMENT_TYPE
    , SBML_CONSTRAINT
    , ...

} SBMLTypeCode_t;
```

# libSBML-5

libSBML core  
(very minor API changes from libSBML-4)

```
typedef enum
{
    SBML_UNKNOWN          = 0
    , SBML_COMPARTMENT     = 1
    , SBML_COMPARTMENT_TYPE = 2
    , SBML_CONSTRAINT      = 3
    , ...
}

} SBMLTypeCode_t;
```

# libSBML-5

libSBML core  
(very minor API changes from libSBML-4)

SBMLTypeCode\_t getTypeCode()

int                    getTypeCode()

# libSBML-5

libSBML core  
(additional API to libSBML-4)

SBML  
Extension

SBasePlugin

SBMLExtension  
Namespaces

SBase  
ExtensionPoint

SBasePlugin  
Creator

SBMLExtension  
Registry

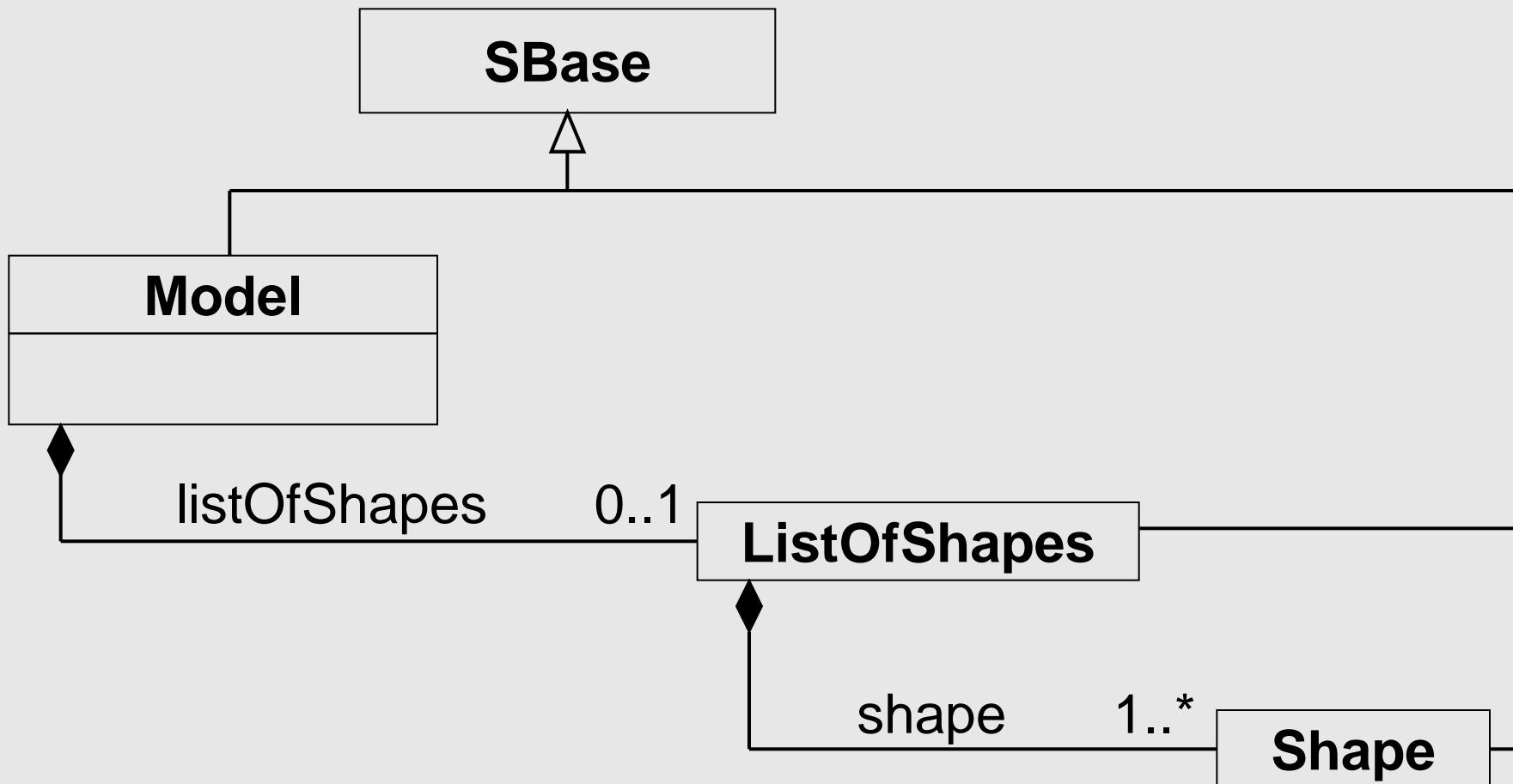
# Implementing a package

“appearance”

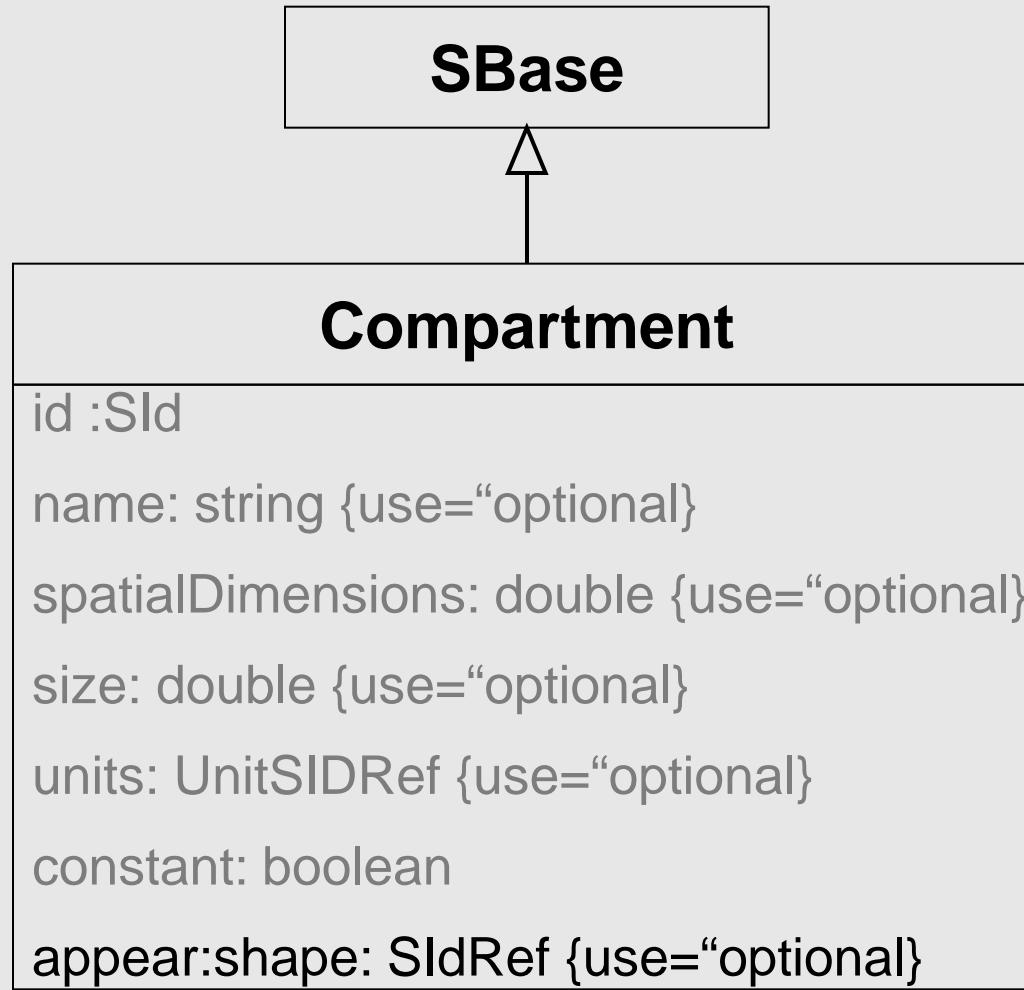
`xmlns:appear="http://www.sbml.org/sbml/level3/version1/appear/version1"`

`appear:required="false"`

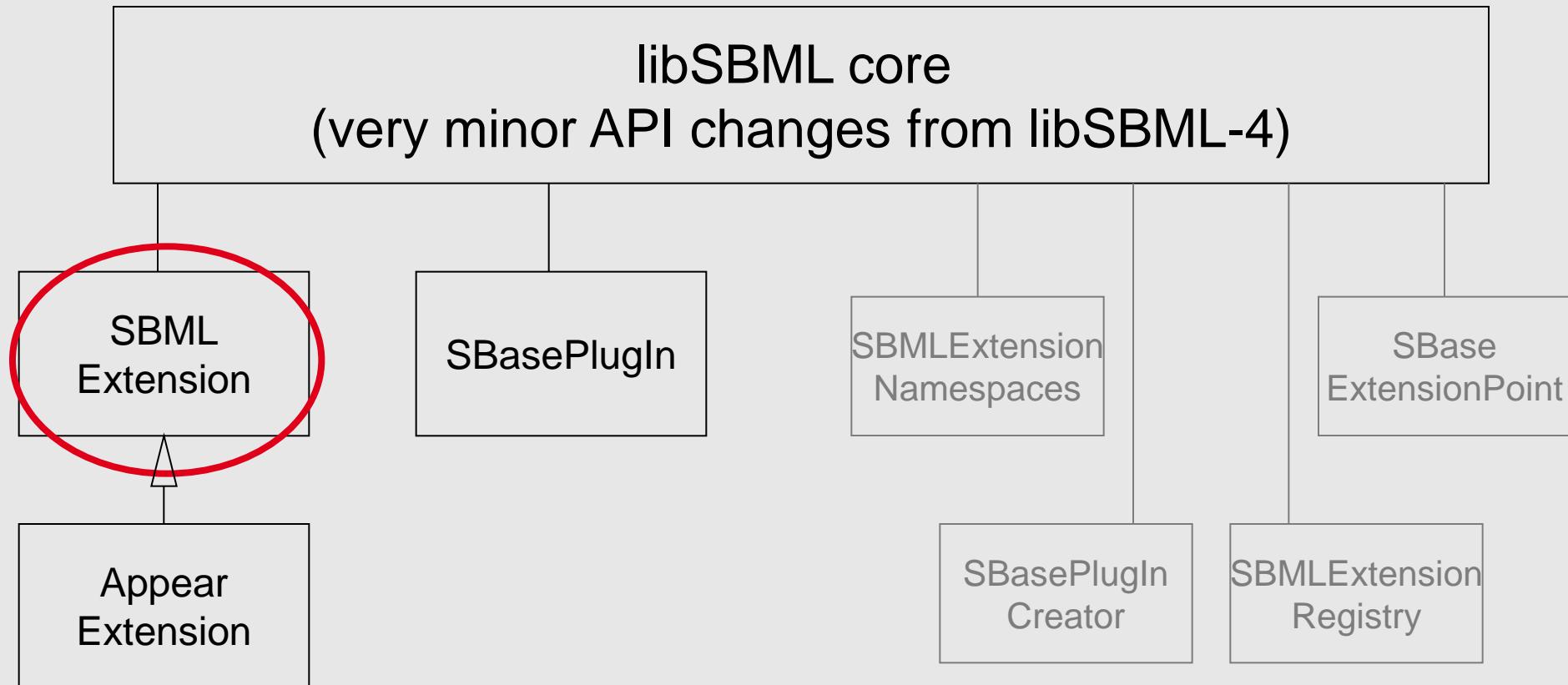
# Implementing a package



# Implementing a package



# Implementing a package



# Implementing a package

## 1. Create the Extension class

```
class AppearExtension : public SBMLExtension  
{  
};
```

# Implementing a package

## 1a. Define necessary variables via functions

```
const std::string& AppearExtension::getPackageName ()  
{  
    static const std::string pkgName = "appear";  
    return pkgName;  
}
```

# Implementing a package

## 1a. Define necessary variables via functions

```
unsigned int AppearExtension::getDefaultLevel()
{
    return 3;
}
unsigned int AppearExtension::getDefaultVersion()
{
    return 1;
}
unsigned int AppearExtension::getDefaultPackageVersion()
{
    return 1;
}
```

# Implementing a package

## 1a. Define necessary variables via functions

```
const std::string& AppearExtension::getXmlNsL3V1V1 ()  
{  
    static const std::string xmlns =  
        "http://www.sbml.org/sbml/level3/version1/appear/version1";  
  
    return xmlns;  
}
```

# Implementing a package

## 1b. Define necessary functions

```
virtual AppearExtension* clone () const;
```

```
virtual SBMLNamespaces* getSBMLExtensionNamespaces(  
                           const std::string &uri) const;
```

```
static void init();
```

# Implementing a package

## 1c. Define the init() function

```
void  
AppearExtension::init()  
{  
    if(SBMLExtensionRegistry::getPackageName().registered(PackageName))  
        return;  
    }  
    AppearExtension appearExtension;  
    std::vector<std::string> packageURIs;  
    packageURIs.push_back(XmlnsL3V1V1);  
    SBaseExtensionPoint sbmlExtPoint("core",SBML_DOCUMENT);  
    SBaseExtensionPoint modelExtPoint("core",SBML_MODEL);  
    SBaseExtensionPoint compExtPoint("core",SBML_COMPARTMENT);  
    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlExtPointCreator(sbmlExtPoint,packageURIs);  
    SBasePluginCreator<AppearModelPlugin, AppearExtension > modelExtPointCreator(modelExtPoint,packageURIs);  
    SBasePluginCreator<AppearCompartmentPlugin, AppearExtension> compExtPointCreator(compExtPoint,packageURIs);  
    appearExtension.addSBasePluginCreator(&sbmlExtPointCreator);  
    appearExtension.addSBasePluginCreator(&modelExtPointCreator);  
    appearExtension.addSBasePluginCreator(&compExtPointCreator);  
    int result = SBMLExtensionRegistry::getInstance().addExtension(&appearExtension);  
  
    if (result != LIBSBML_OPERATION_SUCCESS)  
    {  
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;  
    }  
}
```

# Implementing a package

## 1c. Define the init() function

```
void
AppearExtension::init()
{
    if (SBMLErrorRegistry::getInstance().isRegistered(getPackageName()))
    {
        AppearExtension;
        std::vector<std::string> packageURIs;
        packageURIs.push_back(XmlNs3V1V1);
        SBaseExtensionPoint sbmlDocExtPoint("core",SBML_DOCUMENT);
        SBaseExtensionPoint modelExtPoint("core",SBML_MODEL);
        SBaseExtensionPoint compExtPoint("core",SBML_COMPARTMENT);
        SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlDocPluginCreator(sbmlDocExtPoint,packageURIs);
        SBasePluginCreator<AppearModelPlugin, AppearExtension > modelPluginCreator(modelExtPoint,packageURIs);
        SBasePluginCreator<AppearCompartmentPlugin, AppearExtension> compPluginCreator(compExtPoint,packageURIs);
        appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);
        appearExtension.addSBasePluginCreator(&modelPluginCreator);
        appearExtension.addSBasePluginCreator(&compPluginCreator);

        int result = SBMLErrorRegistry::getInstance().addExtension(&appearExtension);

        if (result != LIBSBML_OPERATION_SUCCESS)
        {
            std::cerr << "[Error] AppearExtension::init() failed." << std::endl;
        }
    }
}
```

# Implementing a package

## 1c. Define the init() function

```
void  
AppearExtension::init()  
{  
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))  
    {  
        return;  
    }  
  
AppearExtension appearExtension;  
    std::vector<std::string> packageURIs;  
    packageURIs.push_back(XmlNs3V1V1);  
    SBaseExtensionPoint sbmlDocExtPoint("core",SBML_DOCUMENT);  
    SBaseExtensionPoint modelExtPoint("core",SBML_MODEL);  
    SBaseExtensionPoint compExtPoint("core",SBML_COMPARTMENT);  
    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlDocPluginCreator(sbmlDocExtPoint,packageURIs);  
    SBasePluginCreator<AppearModelPlugin, AppearExtension > modelPluginCreator(modelExtPoint,packageURIs);  
    SBasePluginCreator<AppearCompartmentPlugin, AppearExtension> compPluginCreator(compExtPoint,packageURIs);  
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);  
    appearExtension.addSBasePluginCreator(&modelPluginCreator);  
    appearExtension.addSBasePluginCreator(&compPluginCreator);  
  
    int result = SBMLErrorRegistry::getInstance().addExtension(&appearExtension);  
  
    if (result != LIBSBML_OPERATION_SUCCESS)  
    {  
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;  
    }  
}
```

# Implementing a package

## 1c. Define the init() function

```
void  
AppearExtension::init()  
{  
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))  
    {  
        return;  
    }  
  
    AppearExtension appearExtension;  
  
    std::vector<std::string> packageURIs;  
    packageURIs.push_back(XmlNsL3V1V1());  
    SBaseExtensionPoint compExtPoint("core", SBML_CCOMPARTMENT);  
    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlDocPluginCreator(sbmlExtPoint, packageURIs);  
    SBasePluginCreator<AppearModelPlugin, AppearExtension> modelPluginCreator(modelExtPoint, packageURIs);  
    SBasePluginCreator<AppearCompartmentPlugin, AppearExtension> compPluginCreator(compExtPoint, packageURIs);  
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);  
    appearExtension.addSBasePluginCreator(&modelPluginCreator);  
    appearExtension.addSBasePluginCreator(&compPluginCreator);  
  
    int result = SBMLErrorRegistry::getInstance().addExtension(&appearExtension);  
  
    if (result != LIBSBML_OPERATION_SUCCESS)  
    {  
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;  
    }  
}
```

might want to include other namespaces

# Implementing a package

## 1c. Define the init() function

```
void
AppearExtension::init()
{
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))
    {
        return;
    }

    AppearExtension appearExtension;
    std::vector<std::string> packageURIs;
    packageURIs.push_back(XmlNsL3V1V1);

    SBaseExtensionPoint sbmlDocExtPoint("core", SBML_DOCUMENT);
    SBaseExtensionPoint modelExtPoint("core", SBML_MODEL);
    SBaseExtensionPoint compExtPoint("core", SBML_COMPARTMENT);

    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> *sbmlDocPluginCreator;
    SBasePluginCreator<AppearCompartmentPlugin, AppearExtension> *modelPluginCreator;
    SBasePluginCreator<AppearModelPlugin, AppearExtension> *compPluginCreator;
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBasePluginCreator(&modelPluginCreator);
    appearExtension.addSBasePluginCreator(&compPluginCreator);

    int result = SBMLErrorRegistry::getInstance().addExtension(&appearExtension);

    if (result != LIBSBML_OPERATION_SUCCESS)
    {
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;
    }
}
```

# Implementing a package

## 1c. Define the init() function

```
void  
AppearExtension::init()  
{  
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))  
    {  
        return;  
    }  
  
    AppearExtension appearExtension;  
    std::vector<std::string> packageURIs;  
    packageURIs.push_back(XmlNsL3V1V1);  
  
    SBaseExtensionPoint sbmlDocExtPoint("core", SBML_DOCUMENT);  
    SBaseExtensionPoint modelExtPoint("core", SBML_MODEL);  
    SBaseExtensionPoint compExtPoint("core", SBML_COMPARTMENT);  
  
    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlDocPluginCreator(appearExtension, "core", "SBML_DOCUMENT");  
    SBasePluginCreator<SBMLModelPlugin, AppearExtension> modelPluginCreator(appearExtension, "core", "SBML_MODEL");  
    SBasePluginCreator<SBMLCompartmentPlugin, AppearExtension> compPluginCreator(appearExtension, "core", "SBML_COMPARTMENT");  
  
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);  
    appearExtension.addSBasePluginCreator(&modelPluginCreator);  
    appearExtension.addSBasePluginCreator(&compPluginCreator);  
  
    int result = SBMLErrorRegistry::getInstance().addExtension(&appearExtension);  
  
    if (result != LIBSBML_OPERATION_SUCCESS)  
    {  
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;  
    }  
}
```

adding  
'required'  
attribute

# Implementing a package

## 1c. Define the init() function

```
void
AppearExtension::init()
{
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))
    {
        return;
    }

    AppearExtension appearExtension;
    std::vector<std::string> packageURIs;
    packageURIs.push_back(XmlNsL3V1V1);

    SBaseExtensionPoint sbmlDocExtPoint("core", SBML_DOCUMENT);
    SBaseExtensionPoint modelExtPoint("core", SBML_MODEL);
    SBaseExtensionPoint compExtPoint("core", SBML_COMPARTMENT);
    SBaseExtensionPoint apExtPoint("core", AppearExtension);
    SBaseExtensionPoint compExtPoint("core", COMPARTMENT);
    SBaseExtensionPoint apExtPoint("core", AppearExtension);
    SBaseExtensionPoint apExtPoint("core", AppearExtension);

    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBasePluginCreator(&modelPluginCreator);
    appearExtension.addSBasePluginCreator(&compPluginCreator);

    int result = SBMLErrorRegistry::getInstance().addExtension(&appearExtension);

    if (result != LIBSBML_OPERATION_SUCCESS)
    {
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;
    }
}
```

adding  
‘ListOfShapes’  
element

# Implementing a package

## 1c. Define the init() function

```
void
AppearExtension::init()
{
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))
    {
        return;
    }

    AppearExtension appearExtension;
    std::vector<std::string> packageURIs;
    packageURIs.push_back(XmlNsL3V1V1);

    SBaseExtensionPoint sbmlDocExtPoint("core", SBML_DOCUMENT);
    SBaseExtensionPoint modelExtPoint("core", SBML_MODEL),
    SBaseExtensionPoint compExtPoint("core", SBML_COMPARTMENT);
    SBaseExtensionPoint apExtPoint("core", AppearExtension);
    SBaseExtensionPoint libExtPoint("core", LIBSBML_OPERATION_SUFFIX);

    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlDocPluginCreator;
    SBasePluginCreator<SBMLModelPlugin, AppearExtension> modelPluginCreator;
    SBasePluginCreator<SBMLCompartmentPlugin, AppearExtension> compPluginCreator;
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBasePluginCreator(&modelPluginCreator);
    appearExtension.addSBasePluginCreator(&compPluginCreator);

    int result = SBMLErrorRegistry::get();
    if (result != LIBSBML_OPERATION_SUCCESS)
    {
        std::cerr << "[Error] AppearExtension";
    }
}
```

adding 'shape' attribute

# Implementing a package

## 1c. Define the init() function

```
void  
AppearExtension::init()  
{  
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))  
    {  
        return;  
    }  
  
    AppearExtension appearExtension;  
    std::vector<std::string> packageURIs;  
    packageURIs.push_back(XmlNsL3V1V1);  
  
    SBaseExtensionPoint sbmlDocExtPoint("core", SBML_DOCUMENT);  
    SBaseExtensionPoint modelExtPoint("core", SBML_MODEL);  
    SBaseExtensionPoint compExtPoint("core", SBML_COMPARTMENT);  
  
    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlDocPluginCreator(sbmldocExtPoint.packageURIs);  
    SBasePluginCreator<SBMLModelPlugin, AppearExtension> modelPluginCreator(modelExtPoint.packageURIs);  
    SBasePluginCreator<SBMLCompartmentPlugin, AppearExtension> compPluginCreator(compExtPoint.packageURIs);  
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);  
    appearExtension.addSBasePluginCreator(&modelPluginCreator);  
    appearExtension.addSBasePluginCreator(&compPluginCreator);  
  
    int result = SBMLErrorRegistry::getInstance().addExtension(&appearExtension);  
  
    if (result != LIBSBML_OPERATION_SUCCESS)  
    {  
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;  
    }  
}
```

could be  
another  
package name

# Implementing a package

## 1c. Define the init() function

```
void
AppearExtension::init()
{
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))
    {
        return;
    }

    AppearExtension* appearExtension = new AppearExtension();
    std::vector<std::string> packageURIs;
    packageURIs.push_back(XmlNs3V1V1);
    SBaseExtensionPoint sbmlDocExtPoint("core", "SBML_DOCUMENT");
    SBaseExtensionPoint modelExtPoint("core", "SBML_MODEL");
    SBaseExtensionPoint compExtPoint("core", "SBML_COMPARTMENT");

    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension>
        sbmlDocPluginCreator(sbmlDocExtPoint, packageURIs);

    SBasePluginCreator<AppearModelPlugin, AppearExtension>
        modelPluginCreator(modelExtPoint, packageURIs);
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBasePluginCreator(&modelPluginCreator);
    appearExtension.addSBasePluginCreator(&compPluginCreator);

    int result = SBMLExtensionRegistry::getInstance().addExtension(&appearExtension);
    if (result != LIBSBML_OPERATION_SUCCESS)
    {
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;
    }
}
```

# Implementing a package

## 1c. Define the init() function

```
void
AppearExtension::init()
{
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))
    {
        return;
    }

    AppearExtension appearExtension;
    std::vector<std::string> packageURIs;
    packageURIs.push_back(XmlNs3V1V1);
    SBaseExtensionPoint sbmlDocExtPoint("core",SBML_DOCUMENT);
    SBaseExtensionPoint modelExtPoint("core",SBML_MODEL);
    SBaseExtensionPoint compExtPoint("core",SBML_COMPARTMENT);
    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlDocPluginCreator(sbmlDocExtPoint,packageURIs);
    SBasePluginCreator<AppearModelPlugin, AppearExtension> modelPluginCreator(modelExtPoint,packageURIs);
    SBasePluginCreator<AppearCompartmentPlugin, AppearExtension> compPluginCreator(compExtPoint,packageURIs);

    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBasePluginCreator(&modelPluginCreator);
    appearExtension.addSBasePluginCreator(&compPluginCreator);

    int result = SBMLExtensionRegistry::getInstance().addExtension(&appearExtension);
    if (result != LIBSBML_OPERATION_SUCCESS)
    {
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;
    }
}
```

# Implementing a package

## 1c. Define the init() function

```
void
AppearExtension::init()
{
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))
    {
        return;
    }

    AppearExtension appearExtension;
    std::vector<std::string> packageURIs;
    packageURIs.push_back(XmlNs3V1V1);
    SBaseExtensionPoint sbmlDocExtPoint("core", SBML_DOCUMENT);
    SBaseExtensionPoint modelExtPoint("core", SBML_MODEL);
    SBaseExtensionPoint compExtPoint("core", SBML_COMPARTMENT);
    SBaseExtensionPoint<SBMLDocumentPlugin, AppearModelPlugin, AppearExtension> sbmlDocPluginCreator(sbmlDocExtPoint, packageURIs);
    SBaseExtensionPoint<AppearModelPlugin, AppearExtension> modelPluginCreator(modelExtPoint, packageURIs);
    SBaseExtensionPoint<AppearCompartmentPlugin, AppearExtension> compPluginCreator(compExtPoint, packageURIs);

    int result = SBMLErrorRegistry::getInstance().addExtension(&appearExtension);
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBasePluginCreator(&modelPluginCreator);
    appearExtension.addSBasePluginCreator(&compPluginCreator);

    if (result != LIBSBML_OPERATION_SUCCESS)
    {
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;
    }
}
```

# Implementing a package

## 1d. Necessary type definitions

```
typedef SBMLExtensionNamespaces<AppearExtension>  
    AppearPkgNamespaces;
```

```
typedef enum  
{  
    SBML_APPEAR_SHAPE = 200  
} SBMLAppearTypeCode_t;
```

SBML\_COMPARTMENT  
SBML\_LAYOUT\_CURVE  
SBML\_GROUPS\_MEMBER

# Implementing a package

## 2. Create the Plugin classes

```
void
AppearExtension::init()
{
    if (SBMLErrorRegistry::getInstance().isRegistered(PackageName))
    {
        return;
    }

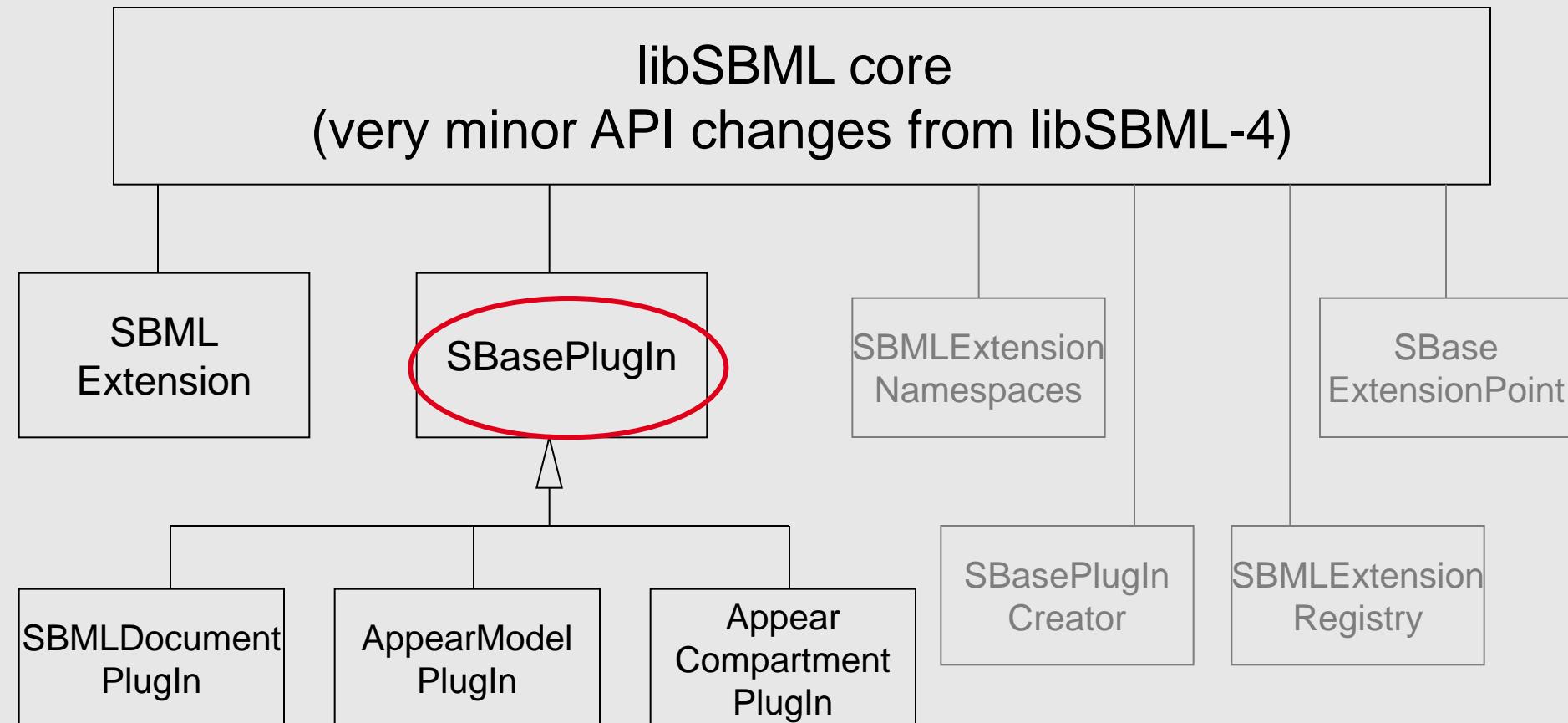
    AppearExtension* appearExtension = new AppearExtension();
    std::vector<std::string> packageURIs;
    packageURIs.push_back(XmlNs3V1V1);
    SBaseExtensionPoint sbmlDocExtPoint("core", "SBML_DOCUMENT");
    SBaseExtensionPoint modelExtPoint("core", "SBML_MODEL");
    SBaseExtensionPoint compExtPoint("core", "SBML_COMPARTMENT");

    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension>
        sbmlDocPluginCreator(sbmlDocExtPoint, packageURIs);

    SBasePluginCreator<AppearModelPlugin, AppearExtension>
        modelPluginCreator(modelExtPoint, packageURIs);
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBasePluginCreator(&modelPluginCreator);
    appearExtension.addSBasePluginCreator(&compPluginCreator);

    int result = SBMLErrorRegistry::getInstance().addExtension(&appearExtension);
    if (result != LIBSBML_OPERATION_SUCCESS)
    {
        std::cerr << "[Error] AppearExtension::init() failed." << std::endl;
    }
}
```

# Implementing a package



# Implementing a package

## 2a. CompartmentPlugIn class

```
class AppearCompartmentPlugIn : public SBasePlugIn
{
public:
    AppearCompartmentPlugin (const std::string &uri, const std::string &prefix, SBMLNamespaces *sbmlNs);
    AppearCompartmentPlugin(const AppearCompartmentPlugin& orig);
    virtual ~AppearCompartmentPlugin ();
    AppearCompartmentPlugin& operator=(const AppearCompartmentPlugin& orig);
    virtual AppearCompartmentPlugin* clone () const;
    virtual void addExpectedAttributes(ExpectedAttributes& attributes);
    virtual void readAttributes (const XMLAttributes& attributes,
                               const ExpectedAttributes& expectedAttributes);
    virtual void writeAttributes (XMLOutputStream& stream) const;
    std::string getShape() const;
    int setShape(std::string value);

protected:
    std::string mShape;
};
```

# Implementing a package

## 2a. CompartmentPlugIn class

```
class AppearCompartmentPlugIn : public SBasePlugIn
{
public:
    AppearCompartmentPlugin (const std::string &uri,
                           const std::string &prefix, SBMLNamespaces *sbmlNs);
    AppearCompartmentPlugin(const AppearCompartmentPlugin& orig);
    virtual ~AppearCompartmentPlugin ();
    AppearCompartmentPlugin& operator=(const
        std::string getShape() const;
        int setShape(std::string value);
    virtual AppearCompartmentPlugin* clone () const;
protected:
    std::string mShape;
};
```

# Implementing a package

## 2a. CompartmentPlugIn class

```
class AppearCompartmentPlugIn : public SBasePlugIn
{
public:
    AppearCompartmentPlugin (const std::string &uri, const std::string &prefix, SBMLNamespaces *sbmlNs);
    AppearCompartmentPlugin(const AppearCompartmentPlugin& orig);
    virtual ~AppearCompartmentPlugin ();
    AppearCompartmentPlugin& operator=(const AppearCompartmentPlugin& orig);
    virtual AppearCompartmentPlugin* clone () const;
    virtual void addExpectedAttributes(ExpectedAttributes& attributes);
    virtual void readAttributes (const XMLAttributes& attributes,
                               const ExpectedAttributes& expectedAttributes);
    virtual void writeAttributes (XMLEOutputStream& stream) const;
    std::string getShape() const;
    int setShape(std::string value);
protected:
    std::string mShape;
};
```

adding ‘shape’  
attribute

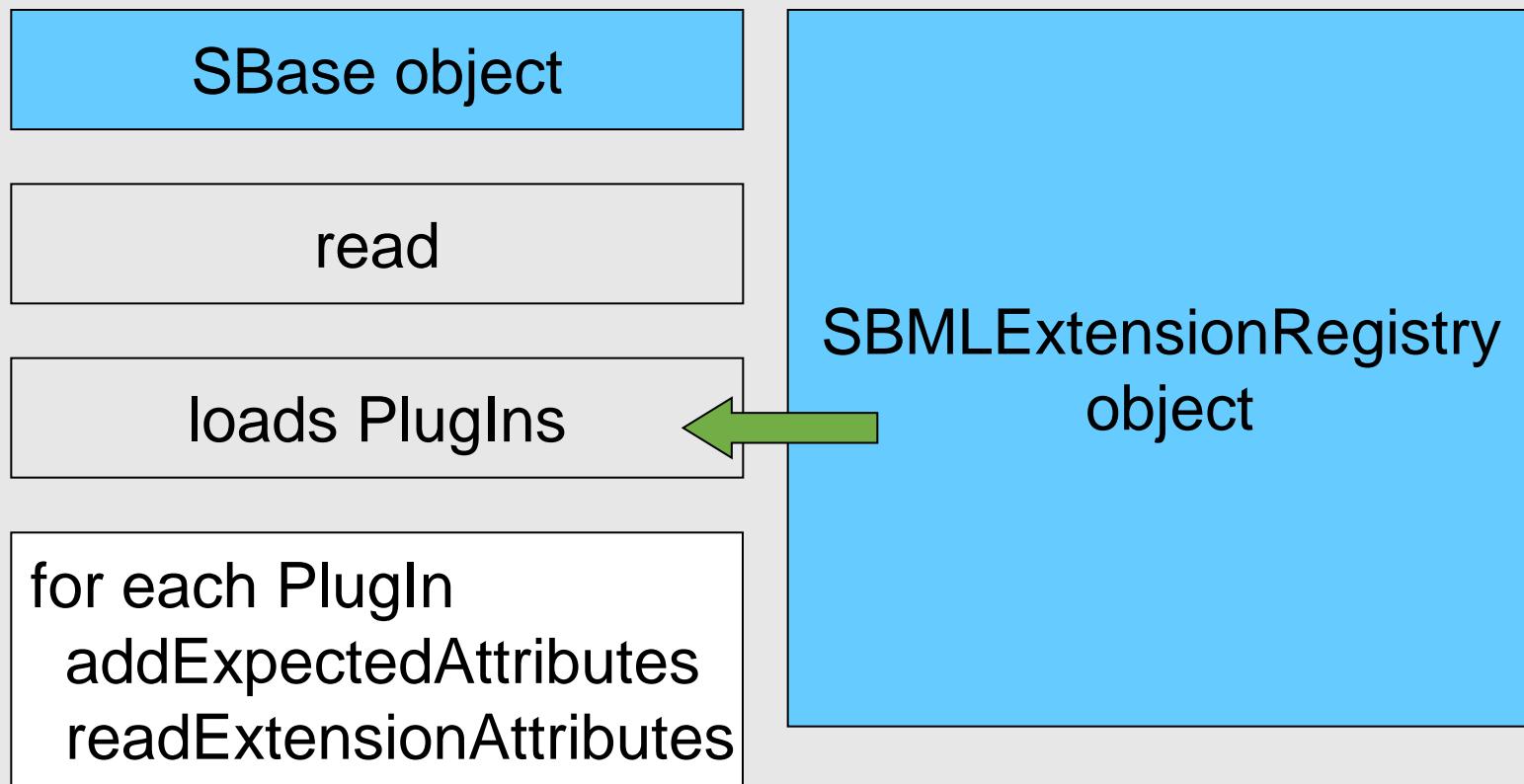
# Implementing a package

## 2a. CompartmentPlugIn class

```
class AppearCompartmentPlugIn : public SBasePlugIn
{
public:
    AppearCompartmentPlugin (const std::string &uri, const std::string &prefix, SBMLNamespaces *sbmlNs);
    AppearCompartmentPlugin(const AppearCompartmentPlugin& orig);
    virtual ~AppearCompartmentPlugin ();
    AppearCompartmentPlugin& operator=(const AppearCompartmentPlugin& orig);
    virtual AppearCompartmentPlugin* clone () const;
    virtual void addExpectedAttributes(ExpectedAttributes& attributes);
    virtual void readAttributes (const XMLAttributes& attributes,
        std::string getShape() const;           const ExpectedAttributes& expectedAttributes);
    virtual void writeAttributes (XMLEOutputStream& stream) const;
protected:
    std::string mShape;
};
```

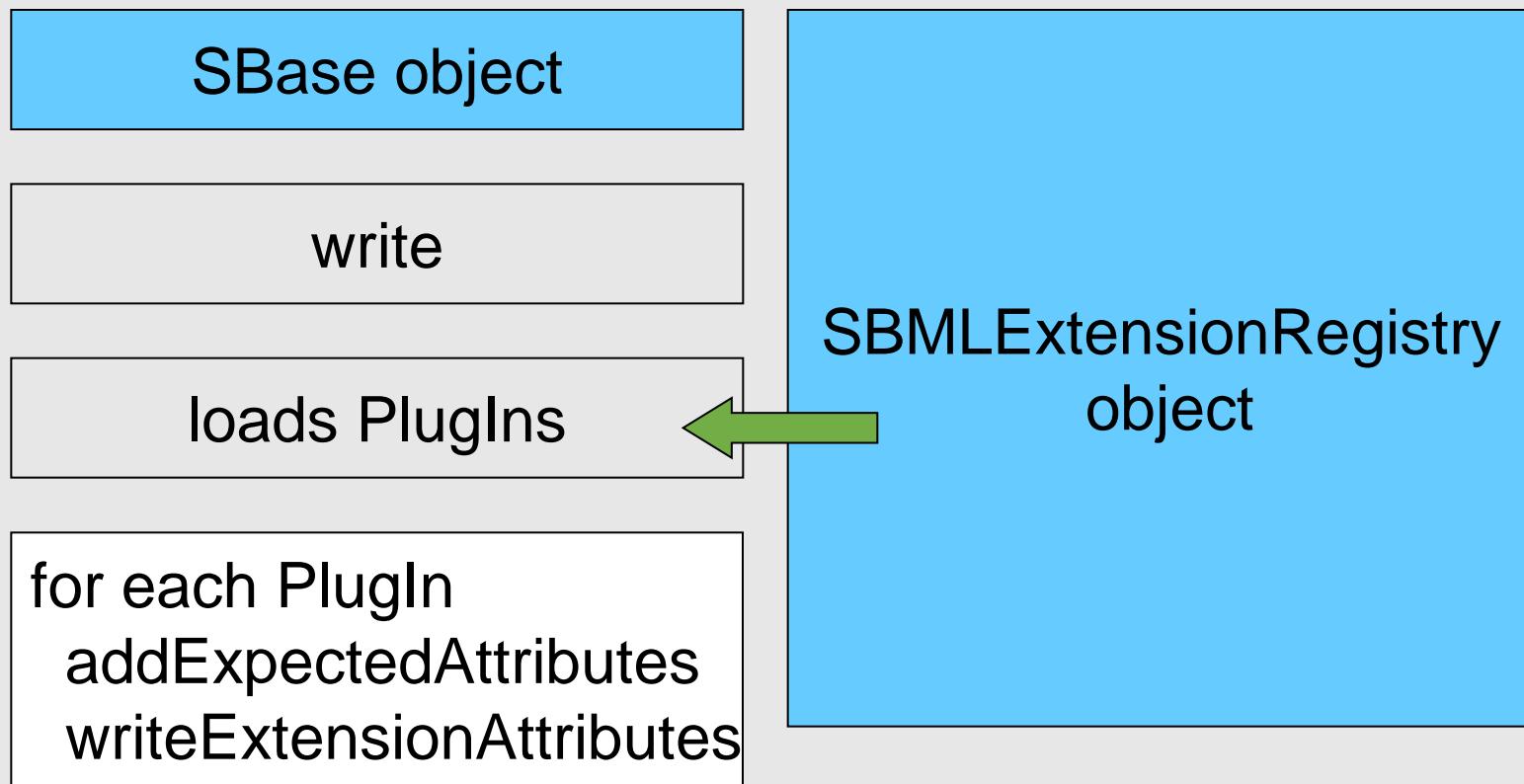
# Implementing a package

## 2a. CompartmentPlugIn class



# Implementing a package

## 2a. CompartmentPlugIn class



# Implementing a package

## 2a. CompartmentPlugIn class

```
class AppearCompartmentPlugIn : public SBasePlugIn
{
public:
    AppearCompartmentPlugin (const std::string &uri, const std::string &prefix, SBMLNamespaces *sbmlNs);
    AppearCompartmentPlugin(const AppearCompartmentPlugin& orig);
    virtual ~AppearCompartmentPlugin ();
    AppearCompartmentPlugin& operator=(const AppearCompartmentPlugin& orig);
    virtual AppearCompartmentPlugin* clone () const;
    virtual void addExpectedAttributes(ExpectedAttributes& attributes);
    virtual void readAttributes (const XMLAttributes& attributes,
        std::string getShape() const;           const ExpectedAttributes& expectedAttributes);
    virtual void writeAttributes (XMLEOutputStream& stream) const;
protected:
    std::string mShape;
};
```

read/write ‘shape’ attribute

# Implementing a package

## 2a. CompartmentPlugIn class

```
class AppearCompartmentPlugIn : public SBasePlugIn
{
public:
    AppearCompartmentPlugin (const std::string &uri, const std::string &prefix, SBMLNamespaces *sbmlNs);
    AppearCompartmentPlugin(const AppearCompartmentPlugin& orig);
    virtual ~AppearCompartmentPlugin ();
    AppearCompartmentPlugin& operator=(const AppearCompartmentPlugin& orig);
    virtual AppearCompartmentPlugin* clone () const;
    virtual void addExpectedAttributes(ExpectedAttributes& attributes);
    virtual void readAttributes (const XMLAttributes& attributes,
                               const ExpectedAttributes& expectedAttributes);
    virtual void writeAttributes (XMLOutputStream& stream) const;
    std::string getShape() const;
    int setShape(std::string value);
protected:
    std::string mShape;
};
```

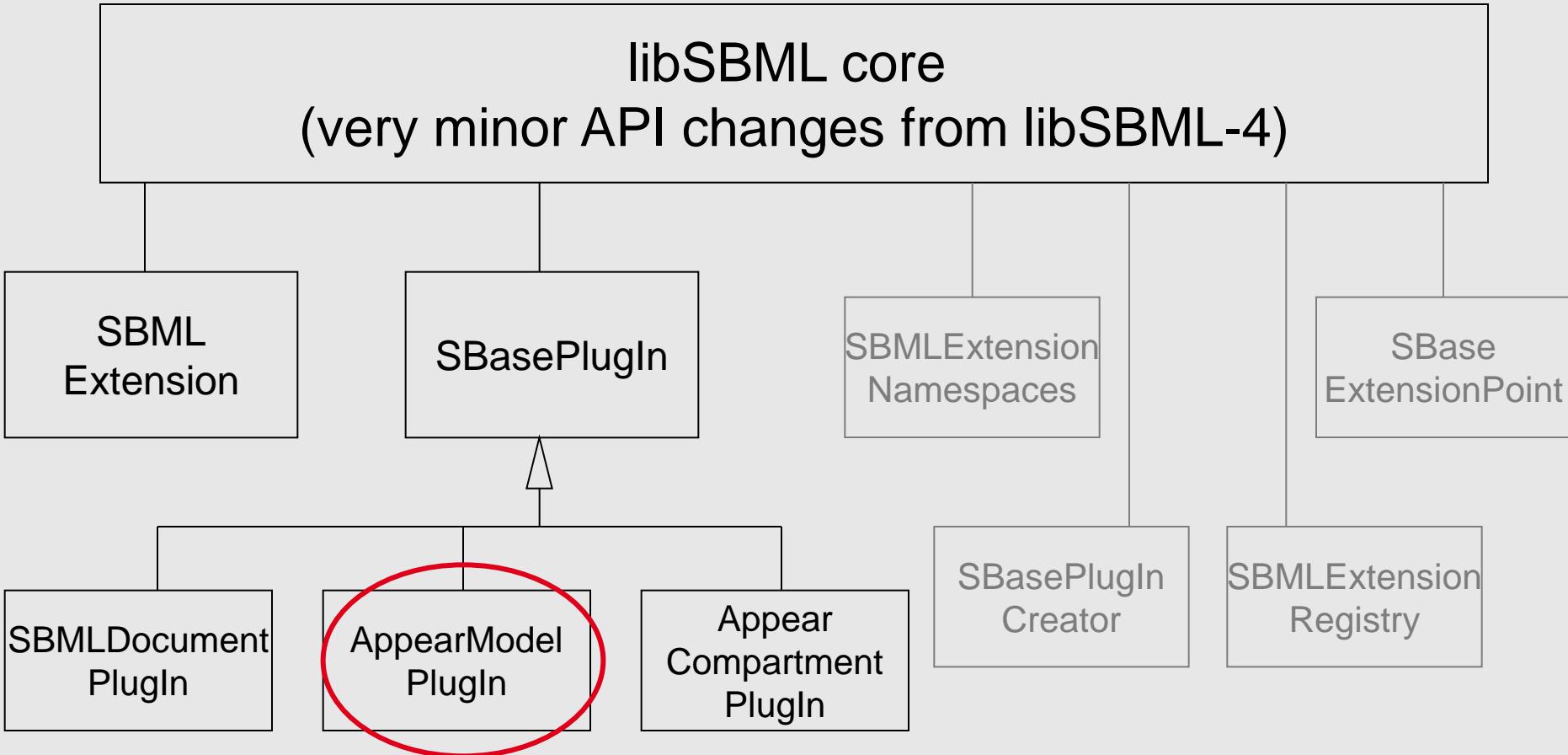
manipulate ‘shape’ attribute

# Implementing a package

## 2a. CompartmentPlugIn class

```
AppearCompartmentPlugin * plugin =  
static_cast<AppearCompartmentPlugin*>(comp->getPlugin("appear");  
  
plugin->setShape("circle");
```

# Implementing a package



# Implementing a package

## 2b. ModelPlugin class

```
class AppearModelPlugin : public SBasePlugin
{
public:
    Base* createObject (XMLInputStream& stream);
    virtual void writeElements (XMLEOutputStream& stream) const;
    const ListOfShapes* getListOfShapes () const;
    Shape* getShape (unsigned int index);
    int addShape (const Shape* Shape);
    Shape* createShape();
    Shape* removeShape (unsigned int n);
    int getNumShapes() const;

protected:
    ListOfShapes mShapes;
};
```

# Implementing a package

## 2b. ModelPlugIn class

```
class LIBSBML_EXTERN AppearModelPlugin : public SBasePlugin
{
public:

    virtual SBase* createObject (XMLInputStream& stream);
    virtual void writeElements (XMLEOutputStream& stream) const;

    const ListOfShapes* getListOfShapes () const;

    Shape* getShape (unsigned int index);

    int addShape (const Shape* Shape);
    Shape* createShape();
    Shape* removeShape (unsigned int n);

protected:
    int getNumShapes() const;

};

ListOfShapes mShapes;
```

adding  
'listOfShapes'  
element

# Implementing a package

## 2b. ModelPlugin class

```
class LIBSBML_EXTERN AppearModelPlugin : public SBasePlugin
{
public:
    virtual SBase* createObject (XMLInputStream& stream);
    virtual void writeElements (XMLEOutputStream& stream) const;
    const ListOfShapes* getListOfShapes () const;
    Shape* getShape (unsigned int index);
    int addShape (const Shape* Shape);
    Shape* createShape();
    Shape* removeShape (unsigned int n);
    int getNumShapes() const;
protected:
    ListOfShapes mShapes;
};
```

create/write  
subelement Shape

# Implementing a package

## 2b. ModelPlugin class

```
class LIBSBML_EXTERN AppearModelPlugin : public SBasePlugin
{
public:
    const ListOfShapes* getListOfShapes () const;
    virtual void writeElements (XMLEOutputStream& stream) const;

    Shape* getShape (unsigned int index);
    int addShape (const Shape* Shape);
    Shape* createShape();
    Shape* removeShape (unsigned int n);

protected:
    int getNumShapes() const;
    ListOfShapes mShapes;
};
```

manipulate  
'listOfShapes' element

# Implementing a package

## 2a. ModelPlugIn class

```
AppearModelPlugin * m_plugin =  
    static_cast<AppearModelPlugin*>  
        (model->getPlugin("appear"));
```

```
Shape *s = m_plugin->createShape();  
s->setId("circle");
```

# Implementing a package

## 3. Using the package

```
int main(int argc,char** argv)
{
    SBMLNamespaces sbmlns(3,1);
    sbmlns.addPkgNamespace("appear", 1);

    SBMLDocument *document = new SBMLDocument(&sbmlns);
    document->setPkgRequired("appear", false);

    Model * m = document->createModel();
    Compartment * c = m->createCompartment();
    c->setId("c1");
    c->setConstant(true);

    AppearModelPlugin * m_plugin = static_cast<AppearModelPlugin*>(m->getPlugin("appear"));

    Shape *s = m_plugin->createShape();
    s->setId("circle");

    AppearCompartmentPlugin * plugin = static_cast<AppearCompartmentPlugin*>(c->getPlugin("appear"));

    plugin->setShape("circle");
    writeSBML(document, 'appear.xml');

    delete document;
}
```

# Implementing a package

## 3. Using the package

```
int main(int argc,char** argv)
{
    SBMLNamespaces sbmlNs(3,1);
    document->setPkgNamespace("appear", false);
    sbmlNs.addPkgNamespace("appear", 1);
    Model * m = document->createModel();
    Compartment * c = m->createCompartment();
    c->setId("c1");
    c->setConstant(true);
    AppearModelPlugin * m_plugin = static_cast<AppearModelPlugin*>(m->getPlugin("appear"));

    Shape *s = m_plugin->createShape();
    s->setId("circle");

    AppearCompartmentPlugin * plugin = static_cast<AppearCompartmentPlugin*>(c->getPlugin("appear"));

    plugin->setShape("circle");
    writeSBML(document, 'appear.xml');

    delete document;
}
```

# Implementing a package

## 3. Using the package

```
int main(int argc,char** argv)
{
    SBMLNamespaces sbmlns(3,1);
    sbmlns.addPkgNamespace("appear", 1,);

    SBMLDocument *document = new SBMLDocument(&sbmlns);
    document->setPkgRequired("appear", false);
    Compartiment * c = m->createCompartiment();
    c->setId("c");
    c->setConstant(true);

    AppearModelPlugin * m_plugin = static_cast<AppearModelPlugin*>(m->getPlugin("appear"));

    Shape *s = m_plugin->createShape();
    s->setId("circle");

    AppearCompartmentPlugin * plugin = static_cast<AppearCompartmentPlugin*>(c->getPlugin("appear"));

    plugin->setShape("circle");
    writeSBML(document, 'appear.xml');

    delete document;
}
```

# Implementing a package

## 3. Using the package

```
int main(int argc,char** argv)
{
    SBMLNamespaces sbmlns(3,1);
    sbmlns.addPkgNamespace("appear", 1);
    SBMLDocument *document = new SBMLDocument(&sbmlns);
    document->setPkgRequired("appear", false);

    Model * m = document->createModel();
    Compartment * c = m->createCompartment();
    c->setId("c");
    AppearModelPlugin * plugin = static_cast<AppearModelPlugin*>(m->getPlugin("appear"));
    c->setConstant(true);
    Shape *s=m->createShape();
    s->setId("circle");
    AppearCompartmentPlugin * plugin = static_cast<AppearCompartmentPlugin*>(c->getPlugin("appear"));
    plugin->setShape("circle");
    writeSBML(document, 'appear.xml');

    delete document;
}
```

# Implementing a package

## 3. Using the package

```
int main(int argc,char** argv)
{
    SBMLNamespaces sbmlns(3,1);
    sbmlns.addPkgNamespace("appear", 1);

    SBMLDocument *document = new SBMLDocument(&sbmlns);
    document->setPkgRequired("appear", false);

    Model * m = document->createModel();
    Compartment * c = m->createCompartment();
    c->id("c");
    c->setConstant(true);

AppearModelPlugin * m_plugin = static_cast<AppearModelPlugin*>
(m->getPlugin("appear"));

    Shape *s = m_plugin->createShape();
s->setId("circle");
    AppearCompartmentPlugin * plugin = static_cast<AppearCompartmentPlugin*>(c->getPlugin("appear"));
    plugin->setShape("circle");
    writeSBML(document, 'appear.xml');

    delete document;
}
```

# Implementing a package

## 3. Using the package

```
int main(int argc,char** argv)
{
    SBMLNamespaces sbmlns(3,1);
    sbmlns.addPkgNamespace("appear", 1);

    SBMLDocument *document = new SBMLDocument(&sbmlns);
    document->setPkgRequired("appear", false);

    Model * m = document->createModel();
    Compartment * c = m->createCompartment();
    c->setId("c1");
    c->setConstant(true);

    AppearModelPlugin * m_plugin = static_cast<AppearModelPlugin*>(m->getPlugin("appear"));

    Shape *s = m_plugin->createShape();
    s->setId("circle");

    AppearCompartmentPlugin * plugin =
        static_cast<AppearCompartmentPlugin*>(c->getPlugin("appear"));
    writeSBML(document, 'appear.xml');

    } // delete document:
        plugin->setShape("circle");
```

# Implementing a package

## 3. Using the package

```
int main(int argc,char** argv)
{
    SBMLNamespaces sbmlns(3,1);
    sbmlns.addPkgNamespace("appear", 1);

    SBMLDocument *document = new SBMLDocument(&sbmlns);
    document->setPkgRequired("appear", false);

    Model * m = document->createModel();
    Compartment * c = m->createCompartment();
    c->setId("c1");
    c->setConstant(true);

    AppearModelPlugin * m_plugin = static_cast<AppearModelPlugin*>(m->getPlugin("appear"));

    Shape *s = m_plugin->createShape();
    s->setId("circle");

    AppearModelPlugin * plugin = static_cast<AppearModelPlugin*>(c->getPlugin("appear"));
    plugin->setShape("circle");

    delete document;
}
```

# Implementing a package

## 3. Using the package

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
      xmlns:appear="http://www.sbml.org/sbml/level3/version1/appear/version1"
      appear:required="false">
  <model>
    <listOfCompartments>
      <compartment id="c" constant="true" appear:shape="circle"/>
    </listOfCompartments>
    <appear:listOfShapes>
      <appear:shape appear:id="circle"/>
    </appear:listOfShapes>
  </model>
</sbml>
```

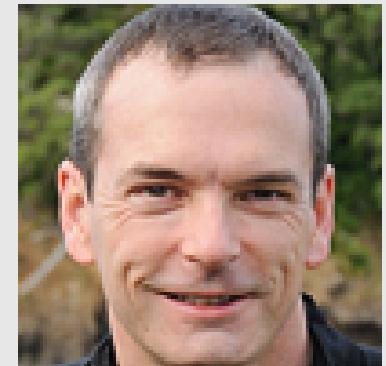
# Acknowledgements



Akiya Jouraku  
Keio, Japan



Frank Bergmann  
Caltech, USA



Mike Hucka  
Caltech, USA

