libSBML-5

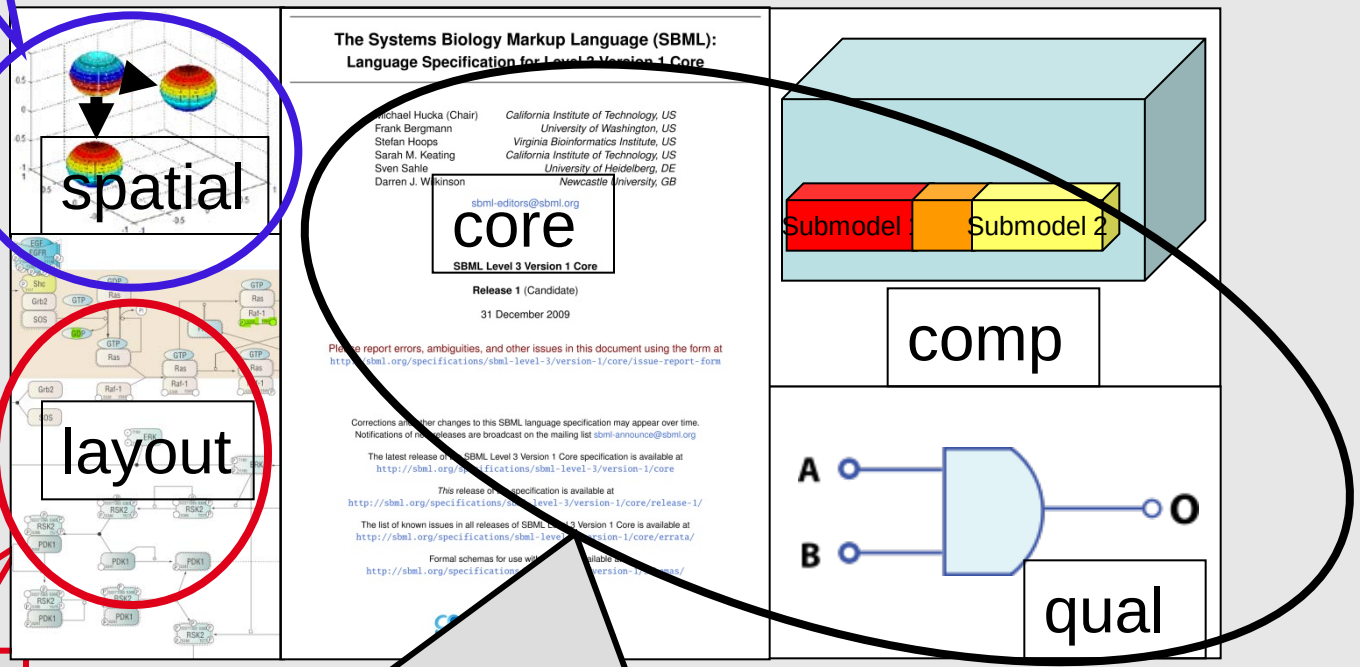
How to write Level 3 packages extensions

Sarah Keating

EMBL-EBI

SBML Level 3

possibly necessary



additional information

mathematically necessary for correct interpretation

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;
```

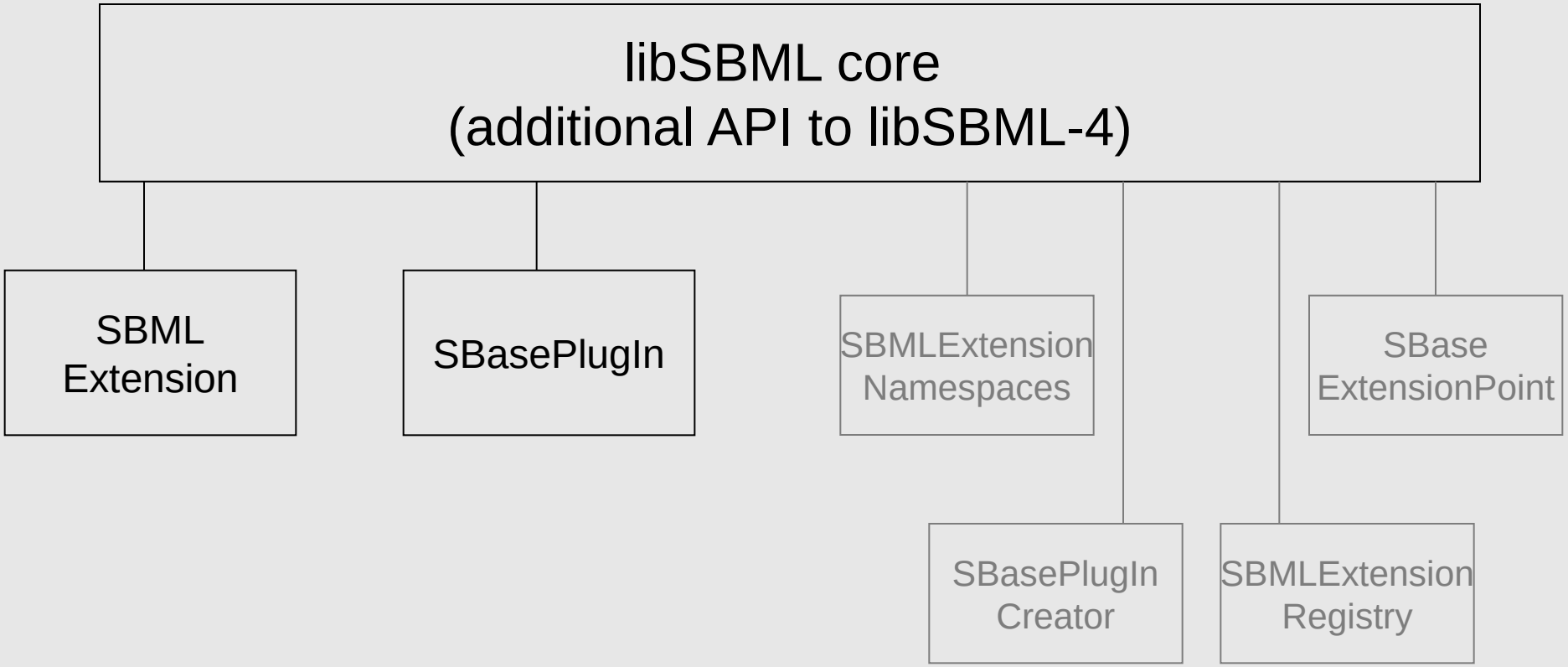
libSBML-5

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

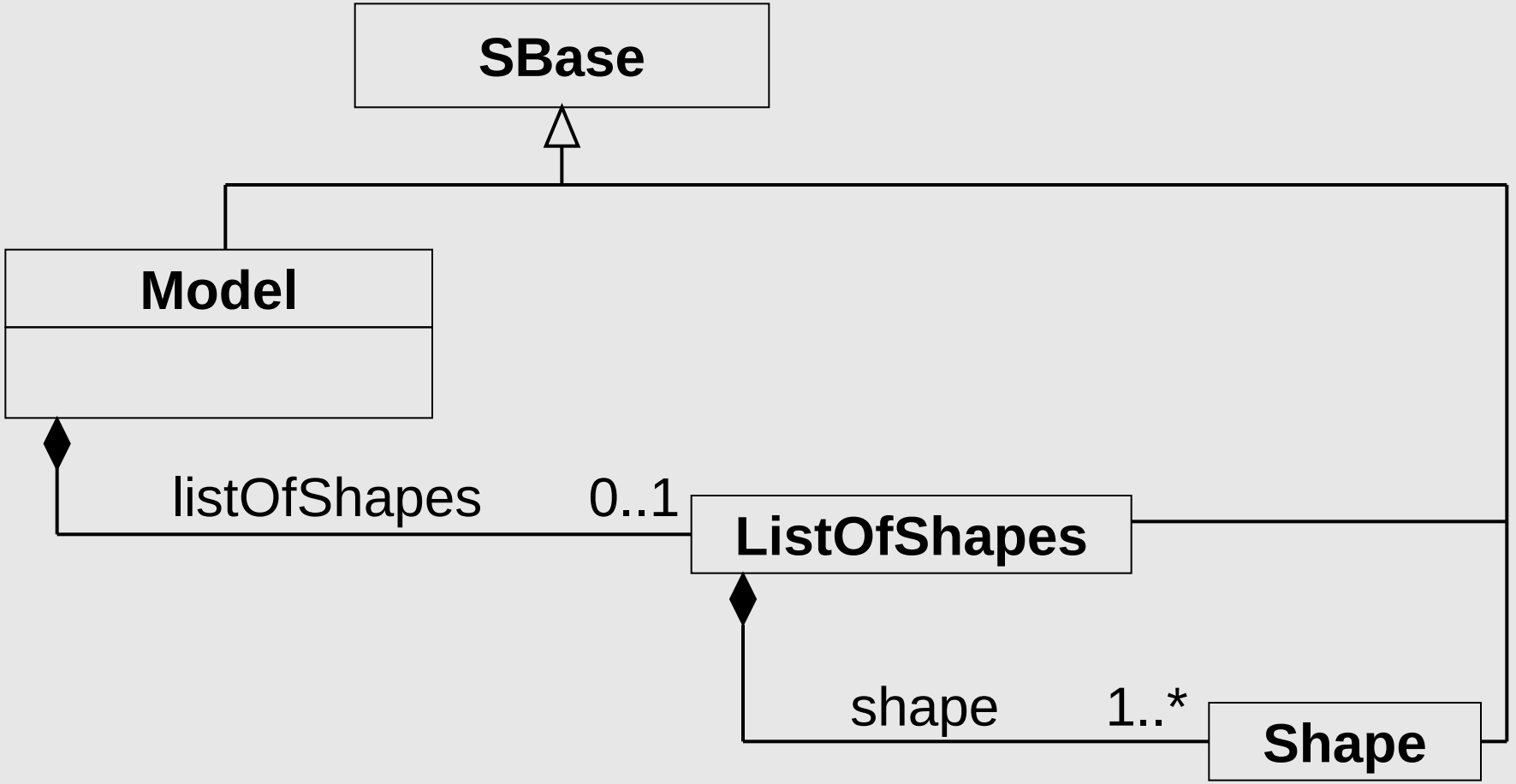
Implementing a package

“appearance”

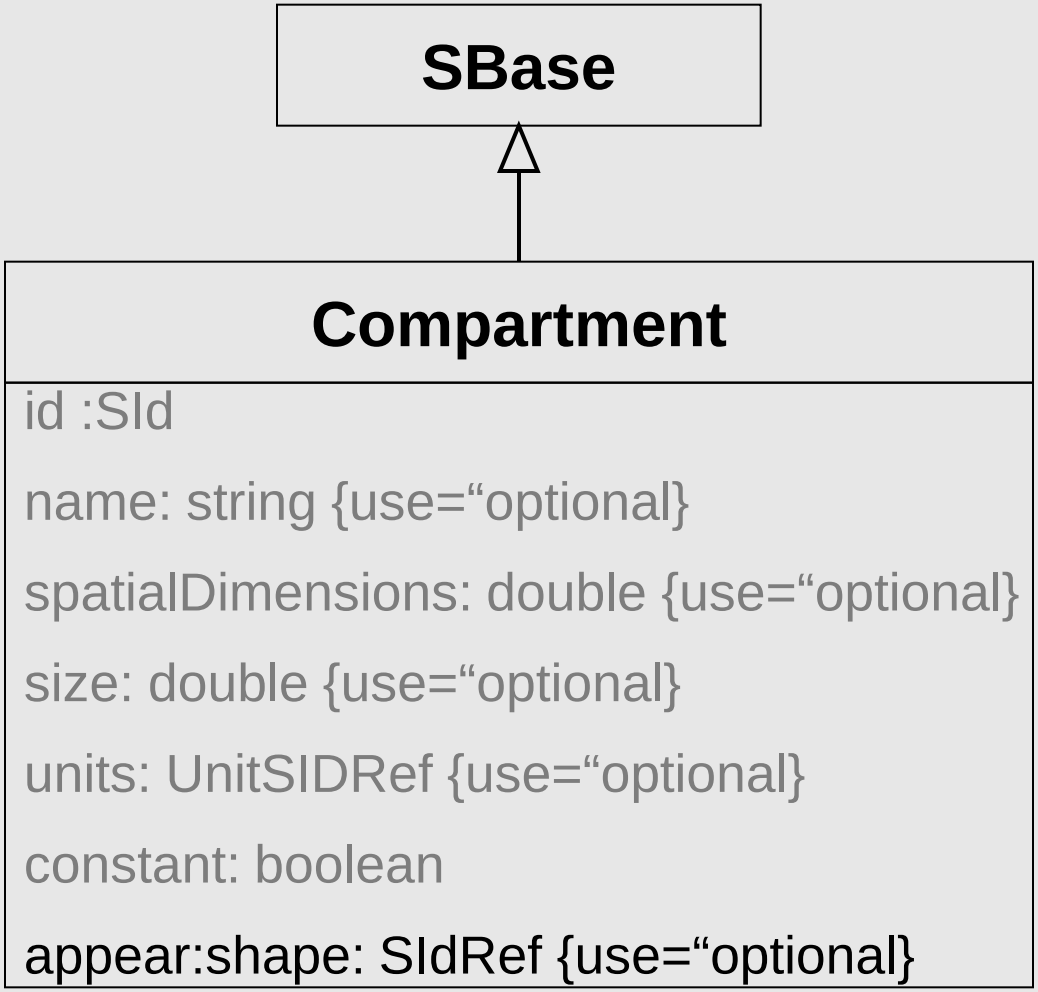
xmlns:appear=“http://www.sbml.org/sbml/level3/version1/appear/version1”

appear:required=“false”

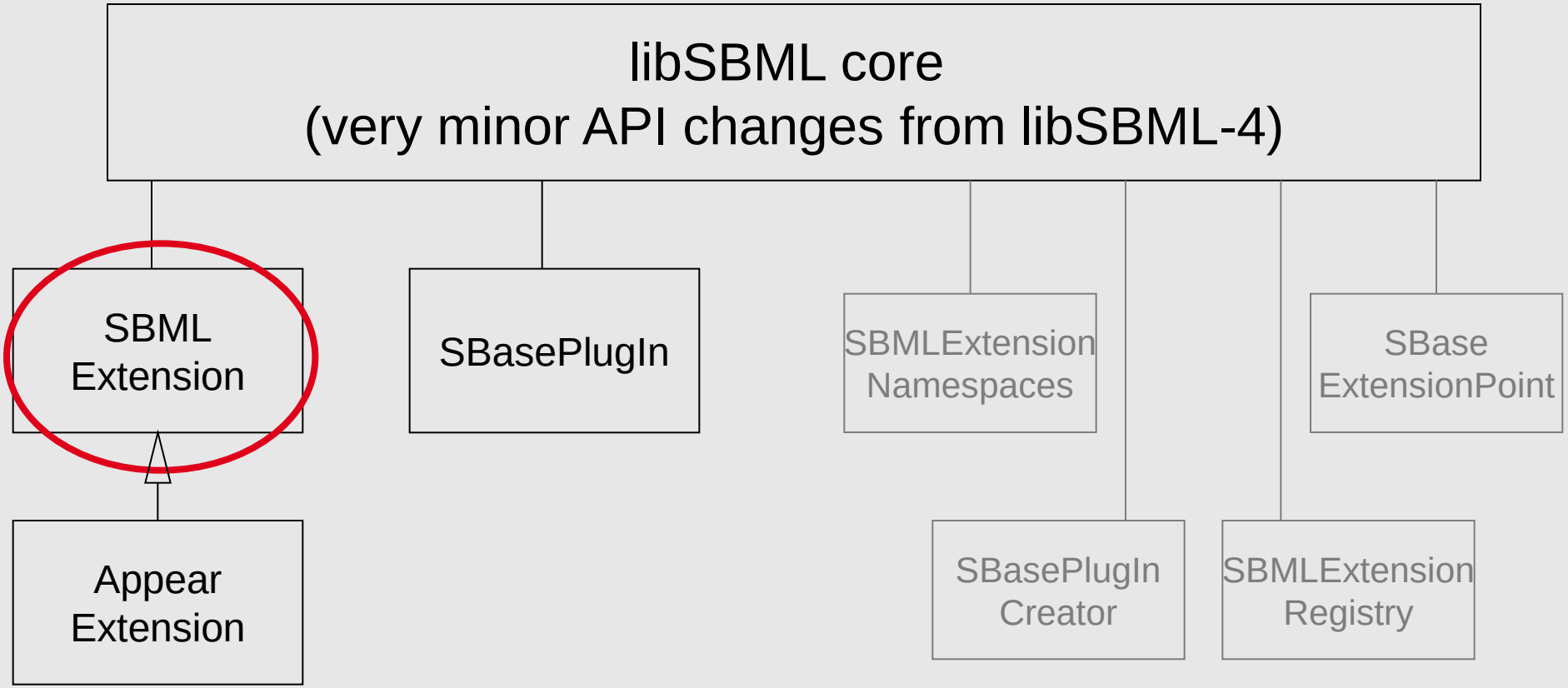
Implementing a package



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::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<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 (SBMLExtensionRegistry::getInstance().isRegistered(getPackageName()))
  {
    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<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 (SBMLExtensionRegistry::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<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 (SBMLExtensionRegistry::getInstance().isRegistered(PackageName))
  {
    return;
  }

  AppearExtension appearExtension;

    std::vector<std::string> packageURIs;
    packageURIs.push_back(getXmlnsL3V1V1());

  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;
  }
}
```

might want to include other namespaces

Implementing a package

1c. Define the init() function

```
void
AppearExtension::init()
{
  if (SBMLExtensionRegistry::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);

  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

adding ‘required’ attribute

```
void
AppearExtension::init()
{
  if (SBMLExtensionRegistry::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<SBMLDocument,AppearSBMLDocumentPlugin> sbmldocPluginCreator(sbmldocExtPoint,packageURIs);
  SBasePluginCreator<Model,AppearModelPlugin> modelPluginCreator(modelExtPoint,packageURIs);
  SBasePluginCreator<Compartment,AppearCompartmentPlugin> 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 (SBMLExtensionRegistry::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<SBMLDocument,AppearExtension> sbmldocPluginCreator(&sbmldocExtPoint,packageURIs);
  SBasePluginCreator<Model,AppearExtension> modelPluginCreator(&modelExtPoint,packageURIs);
  SBasePluginCreator<Compartment,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;
  }
}
```

adding ‘ListOfShapes’ element

Implementing a package

1c. Define the init() function

```
void
AppearExtension::init()
{
  if (SBMLExtensionRegistry::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);

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

  int result = SBMLExtensionRegistry::getI

  if (result != LIBSBML_OPERATION_SU
  {
    std::cerr << "[Error] AppearExtension
  }
}
```

adding ‘shape’ attribute

Implementing a package

1c. Define the init() function

```
void
AppearExtension::init()
{
  if (SBMLExtensionRegistry::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);

  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;
  }
}
```

could be another package name

Implementing a package

1c. Define the init() function

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

  SBasePluginCreator<SBMLDocumentPlugin, AppearExtension>
                                  sbmldocPluginCreator(sbmldocExtPoint,packageURIs);

  SBasePluginCreator<AppearModelPlugin, AppearExtension>
                                  modelPluginCreator(modelExtPoint,packageURIs);

  SBasePluginCreator<AppearCompartmentPlugin,  AppearExtension>
                                  compPluginCreator(compExtPoint,packageURIs);
}
```

Implementing a package

1c. Define the init() function

```
void
AppearExtension::init()
{
  if (SBMLExtensionRegistry::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<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 (SBMLExtensionRegistry::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<Compartment,AppearExtension> modelPluginCreator(modelExtPoint,packageURIs);
  SBasePluginCreator<Compartment,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

1d. Necessary type definitions

```
typedef SBMLExtensionNamespaces<AppearExtension>
                                    AppearPkgNamespaces;

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

SBML_COMPARTMENT

SBML_LAYOUT_CURVE

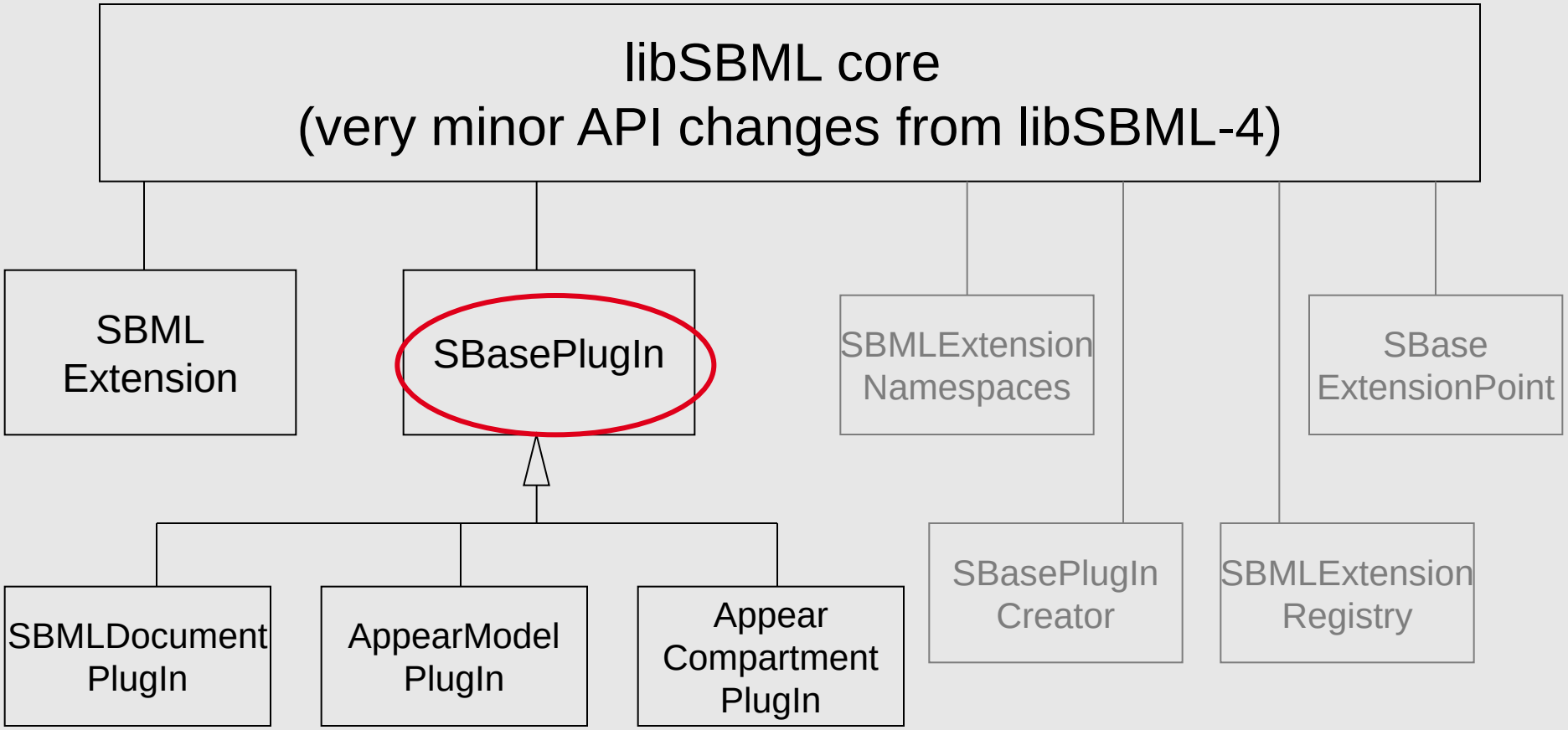
SBML_GROUPS_MEMBER

Implementing a package

2. Create the PlugIn classes

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

  SBasePluginCreator<SBMLDocumentPlugin, AppearExtension>
                    sbmldocPluginCreator(sbmldocExtPoint,packageURIs);

  SBasePluginCreator<AppearModelPlugin, AppearExtension>
                    modelPluginCreator(modelExtPoint,packageURIs);

  SBasePluginCreator<AppearCompartmentPlugin,  AppearExtension>
                    compPluginCreator(compExtPoint,packageURIs);

  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
                                       AppearCompartmentPlugin& orig);
    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 (XMLOutputStream& stream) const;
        std::string getShape() const;
        int setShape(std::string value);
protected:

    std::string mShape;
};
```

adding ‘shape’ attribute

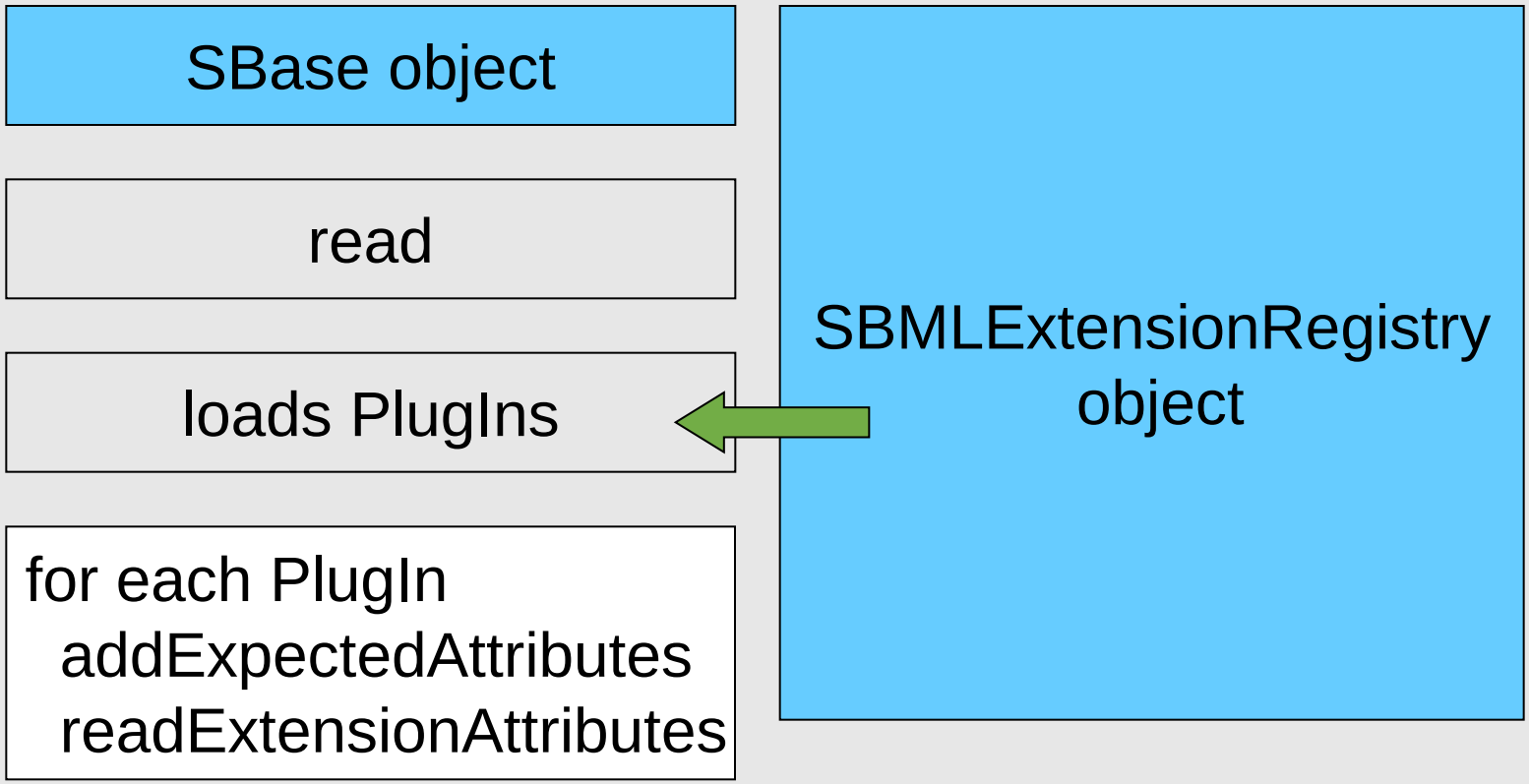
Implementing a package

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,
                                     const ExpectedAttributes& expectedAttributes);
        std::string getShape() const;
        int setShape(std::string value);
        virtual void writeAttributes (XMLOutputStream& stream) const;
protected:

    std::string mShape;
};
```

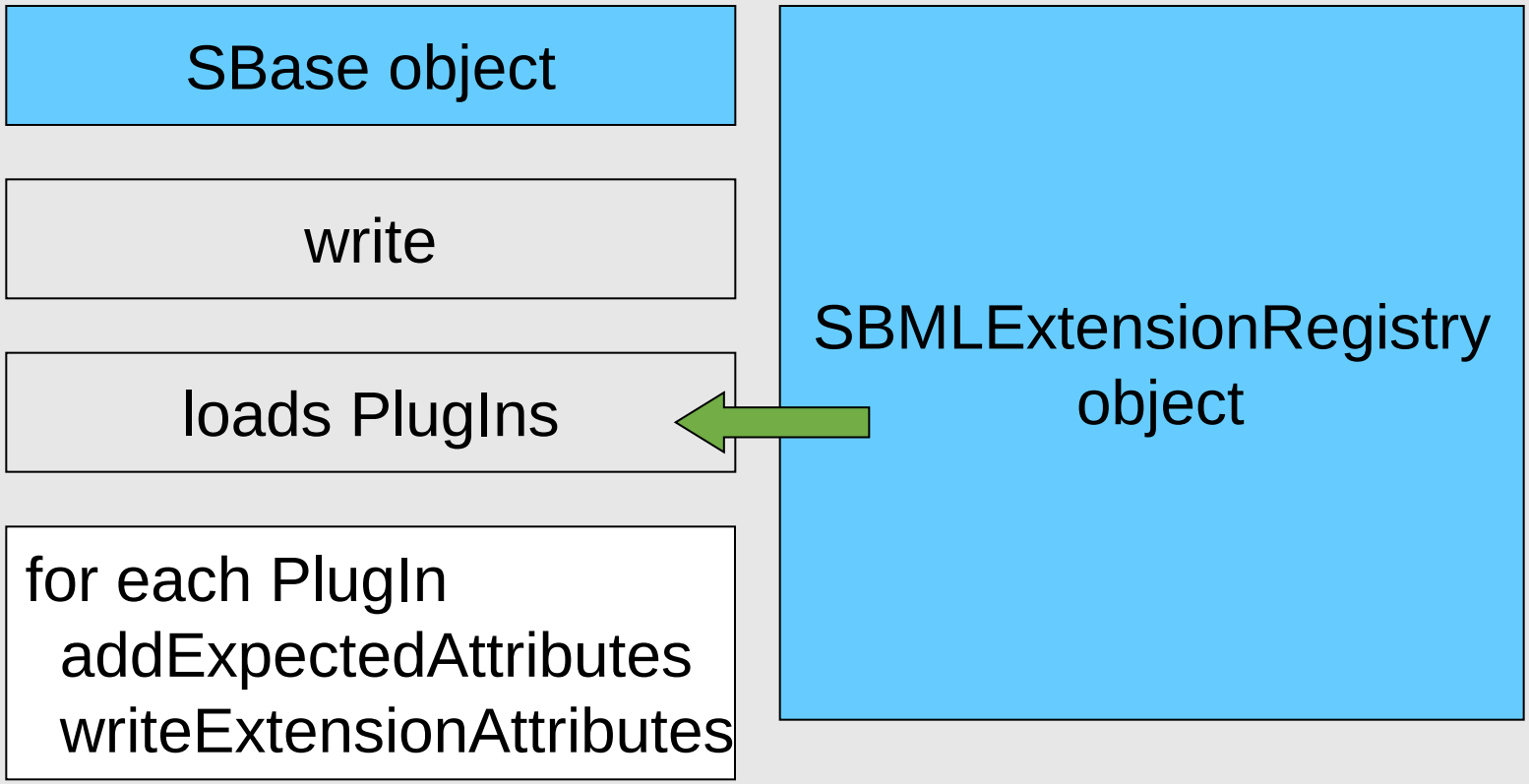
Implementing a package

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);
        int setShape(std::string value);
    virtual void writeAttributes (XMLOutputStream& 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;
};
```

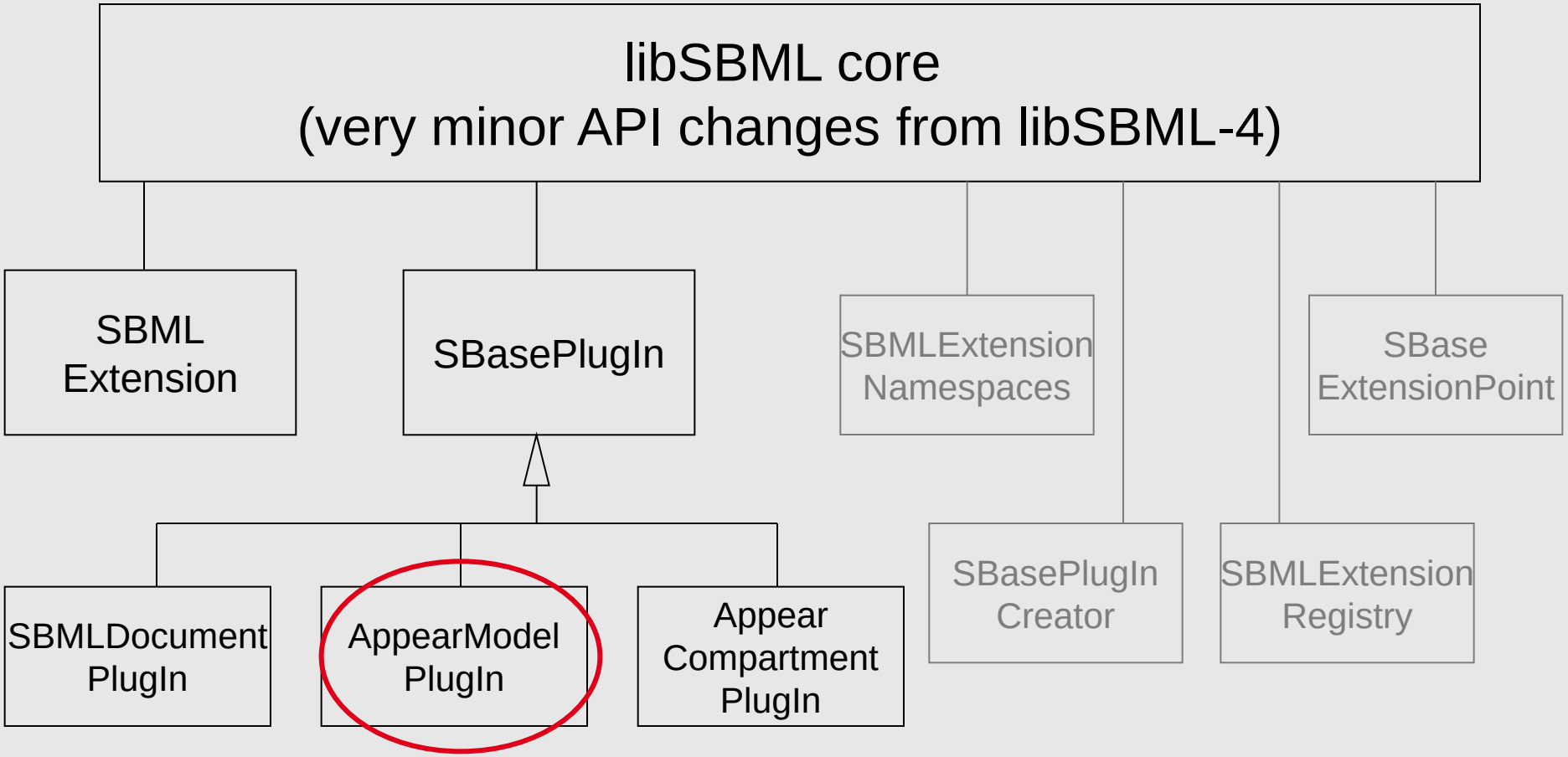
manipulate ‘shape’ attribute

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:
   SBase* createObject (XMLInputStream& stream);
   virtual void writeElements (XMLOutputStream& 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 (XMLOutputStream& 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 (XMLOutputStream& 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 mSha
};
```

create/write
subelement Shape

Implementing a package

2b. ModelPlugIn class

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

   virtual void readAttributes (const XMLAttributes& attributes,
   virtual void writeElements (XMLOutputStream& 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;
};
```

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

  AppearModelPlugin * mplugin = 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");
  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");
  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("c");
  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");

  writeSBML(document, 'appear.xml');

  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

National Institute of General Medical Sciences