

# Which libSBML ?



libSBML-3.4.1

libSBML-4.0.1

libSBML-4.1.0

libSBML-5.0.0

# libSBML-3.4.1



New users should NOT use

Current users PLEASE upgrade



# libSBML-3.4.1

- Bug fixes in later versions - validation
- Code not supported
- Does not support SBML L3

# libSBML-4.0.1

- API changes from 3.4.1
- Fully stable release
- Does not support SBML L3



# libSBML-4.1.0-b3

- Supports SBML L3 core EXCEPT
  - unit validation of L3 models
  - translation of L3 models to L2
  - documentation
- Build issues with more recent OS(es)
- WILL BE A STABLE RELEASE ASAP






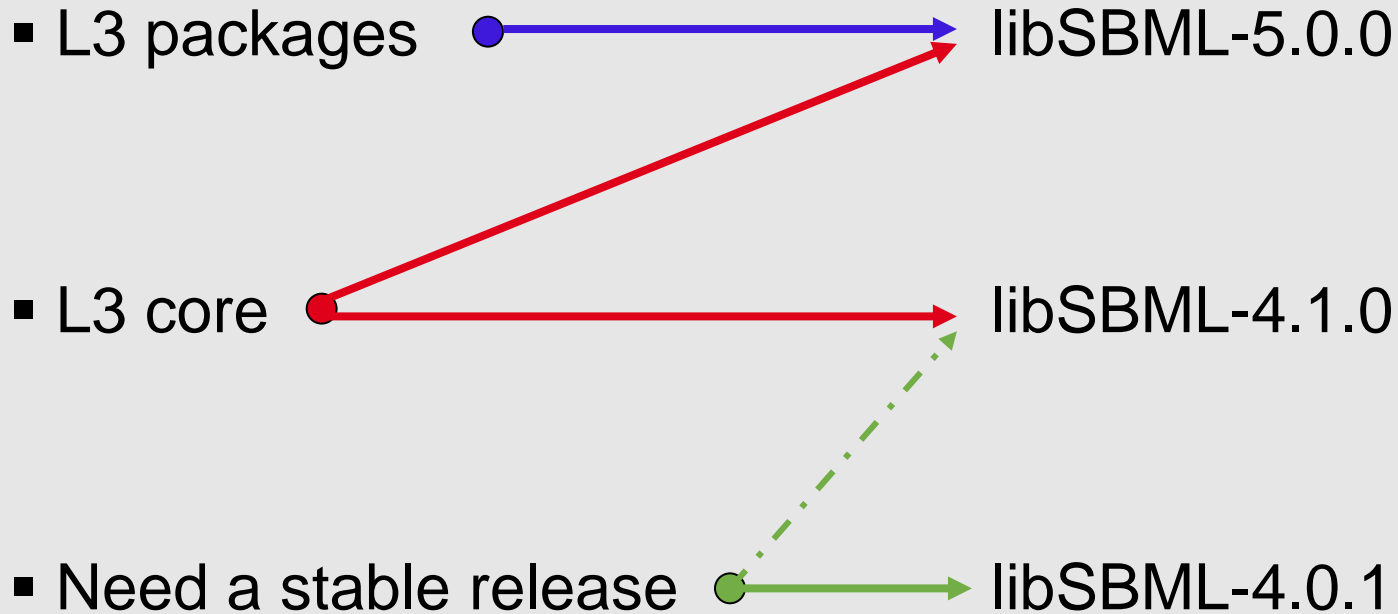


# libSBML-5.0.0-a2

- Identical to 4.1.0
  - ONE API change
  - adds support for packages as extensions in C++
- WILL BE THE ONLY SUPPORTED VERSION

EVENTUALLY

# So which libSBML ??

- L3 packages   libSBML-5.0.0
  - L3 core   libSBML-4.1.0
  - Need a stable release   libSBML-4.0.1
- 

# libSBML-5

## How to write Level 3 packages extensions

Sarah Keating

(on behalf of Akiya Jouraku)



# SBML Level 3

possibly necessary

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  
**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 other changes to this SBML language specification may appear over time. Notifications of these changes 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 validators are available at <http://sbml.org/specifications/sbml-level-3/version-1/core/schemas/>

**spatial**

**comp**

**qual**

**layout**

Submodel 1 Submodel 2

A B O

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;
```

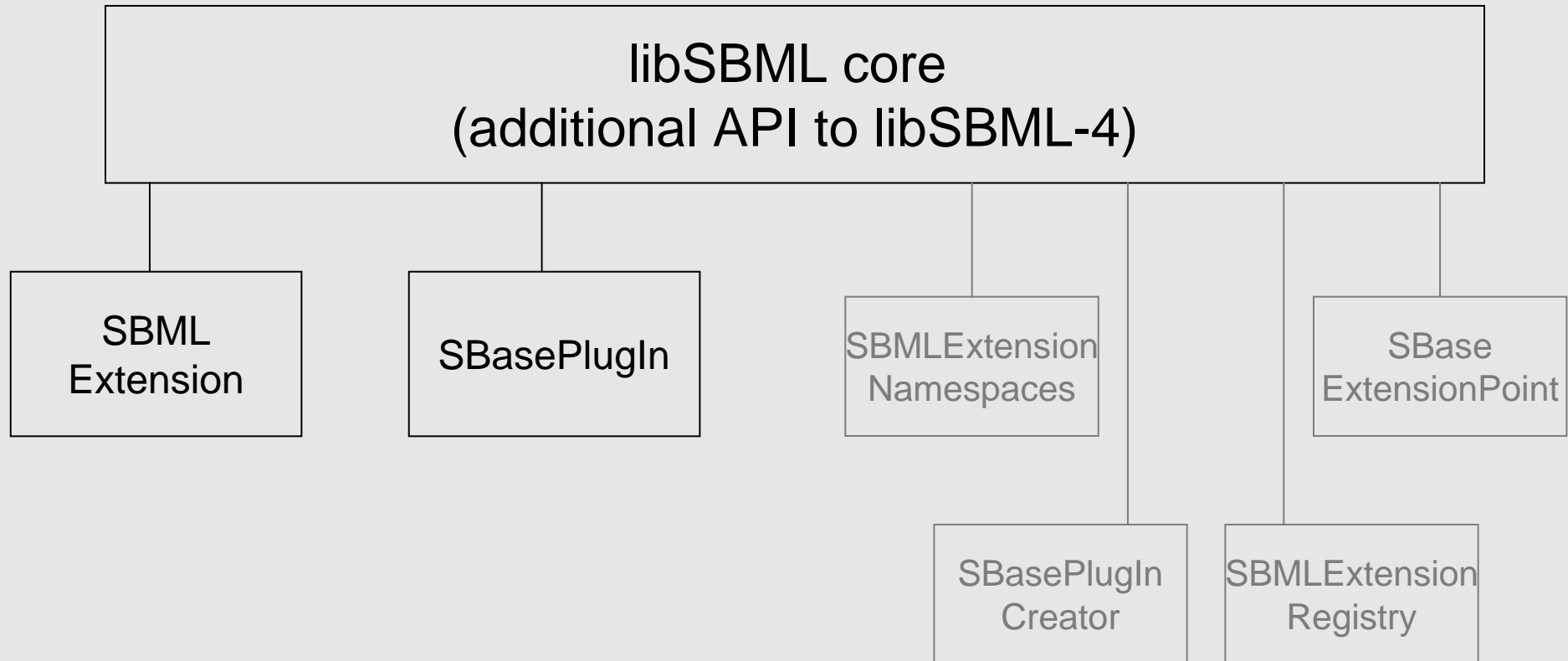
# libSBML-5

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

SBMLTypeCode\_t getTypeCode()

int                   getTypeCode()

# libSBML-5



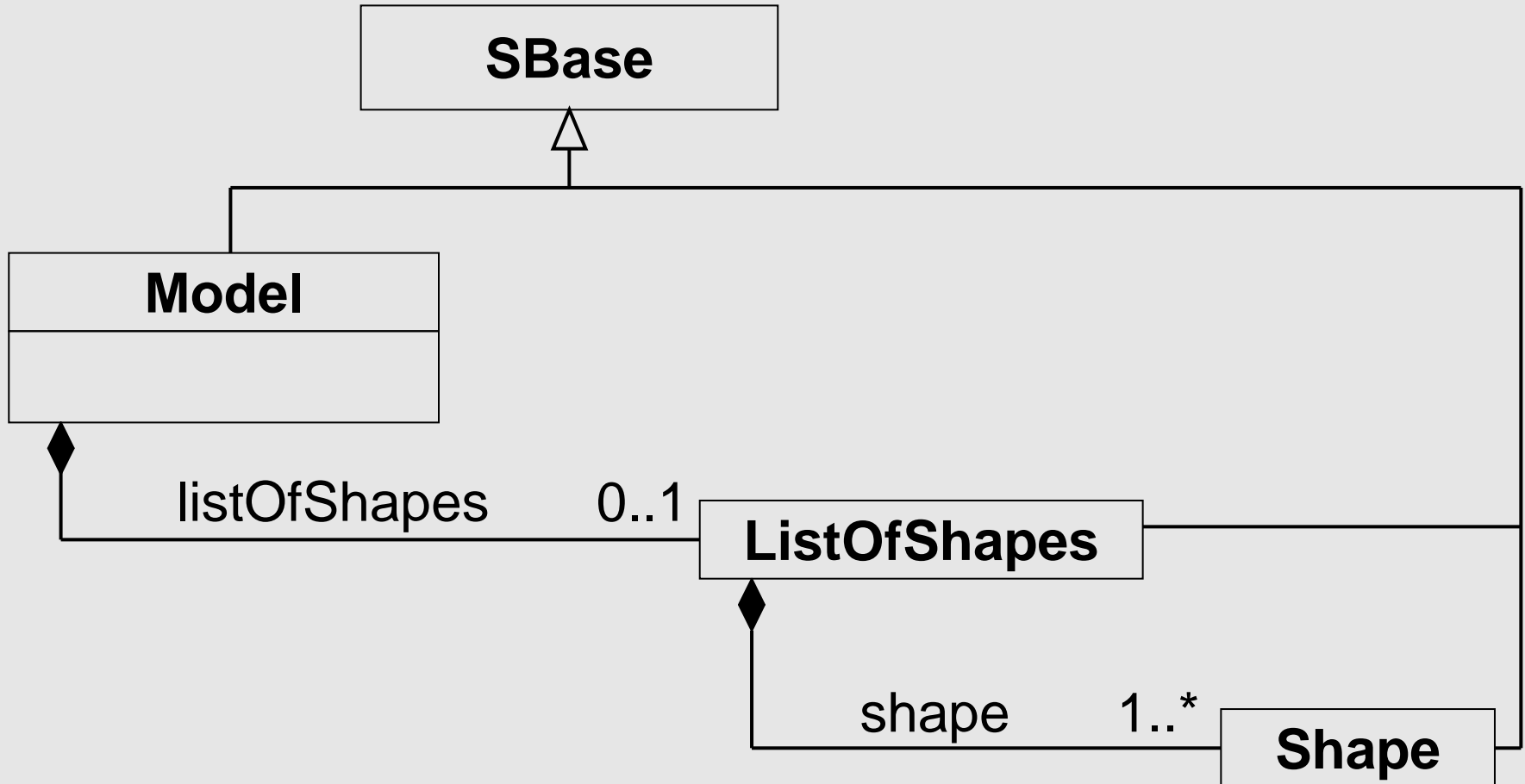
# Implementing a package

“appearance”

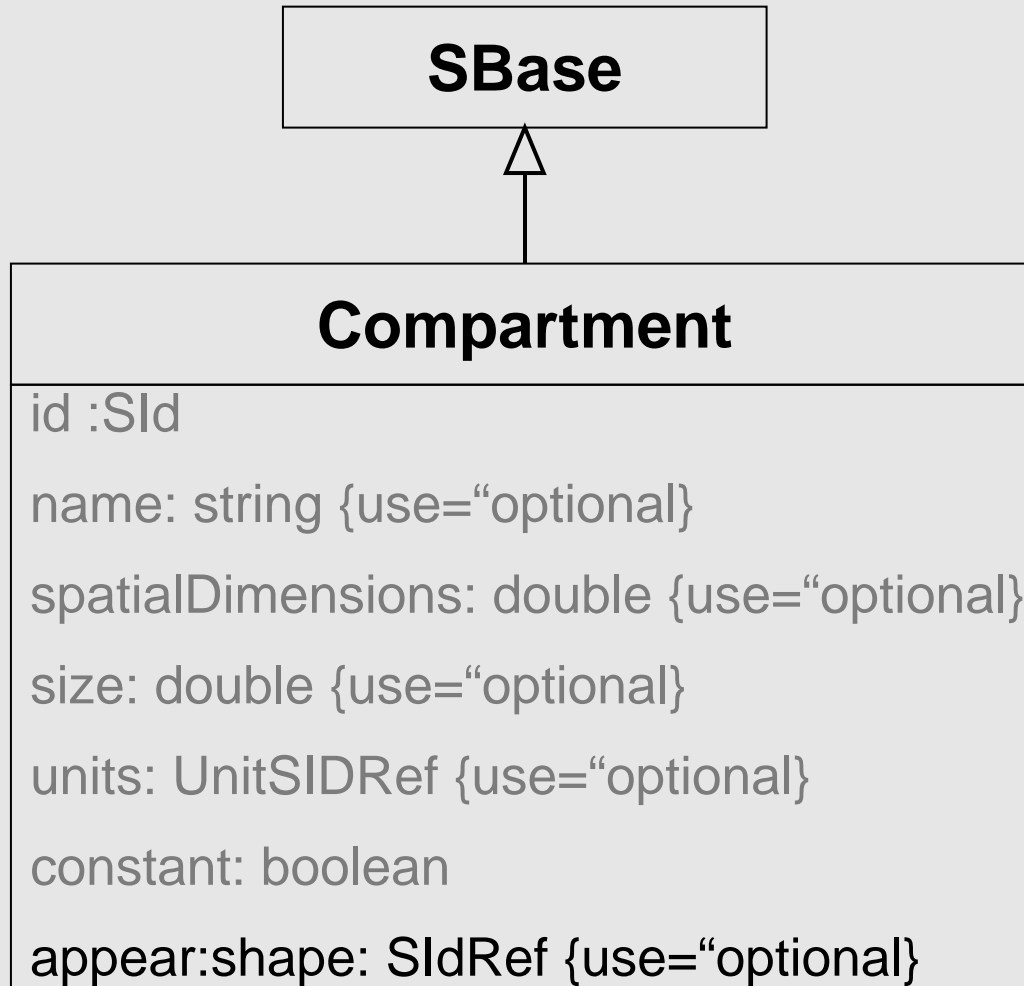
`xmlns:appear="http://www.sbml.org/sbml/level3/version1/appear/version1"`

`appear:required="false"`

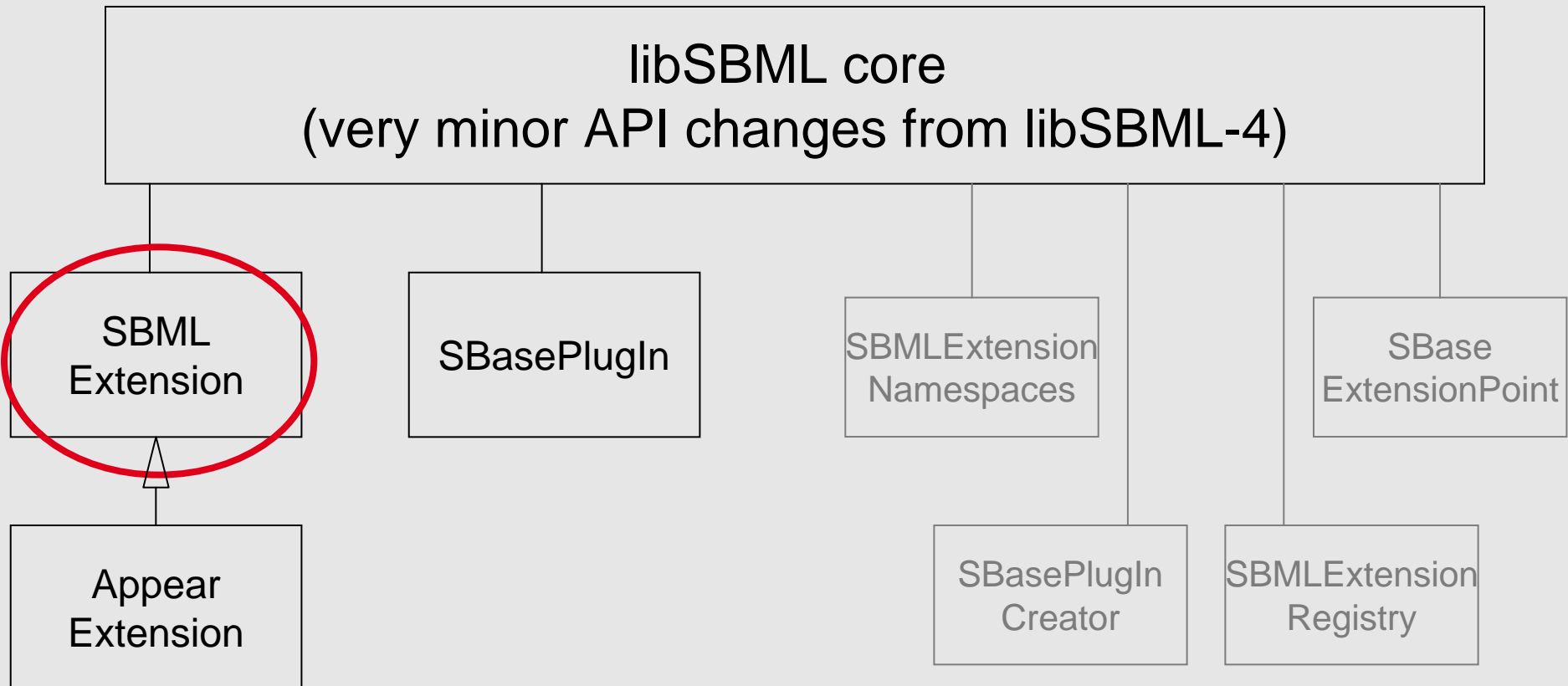
# Implementing a package



# 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

```
const std::string AppearExtension::PackageName = "appear"

const unsigned int AppearExtension ::DefaultLevel           = 3;
const unsigned int AppearExtension ::DefaultVersion        = 1;
const unsigned int AppearExtension ::DefaultPackageVersion = 1;

const std::string AppearExtension::XmlnsL3V1V1 =
    "http://www.sbml.org/sbml/level3/version1/appear/version1";
```

# Implementing a package

## 1b. Define necessary functions

```
virtual AppearExtension* clone () const;  
  
virtual unsigned int getPackageVersion(const std::string &uri) const;  
  
virtual SBMLNamespaces* getSBMLExtensionNamespaces(  
                                                    const std::string &uri) const;  
  
static void init();
```

# Implementing a package

## 1c. Define the init() function

```
void  
  
AppearExtension::init()  
{  
    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;
    }
    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(XmlInsl3V1V1);

    SBBaseExtensionPoint docExtPoint("core", SBML_DOCUMENT);
    SBBaseExtensionPoint compExtPoint("core", SBML_COMPARTMENT);
    SBBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlDocPluginCreator(sbmlDocExtPoint, packageURIs);
    SBBasePluginCreator<AppearModelPlugin, AppearExtension > modelPluginCreator(modelExtPoint, packageURIs);
    SBBasePluginCreator<AppearCompartmentPlugin, AppearExtension> compPluginCreator(compExtPoint, packageURIs);
    appearExtension.addSBBasePluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBBasePluginCreator(&modelPluginCreator);
    appearExtension.addSBBasePluginCreator(&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);
    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 = 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(s, packageURIs);
    SBasePluginCreator<SBMLModelPlugin, AppearExtension> modelPluginCreator(s, packageURIs);
    SBasePluginCreator<SBMLCompartmentPlugin, AppearExtension> compPluginCreator(s, 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  
'required'  
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);  
  
    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension> sbmlDocPluginCreator("sbmlDocExt", packageURIs);  
    SBasePluginCreator<SBMLModelPlugin, AppearExtension> modelPluginCreator("modelExt", packageURIs);  
    SBasePluginCreator<SBMLCompartmentPlugin, AppearExtension> compPluginCreator("compExt", 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);

    SBasePluginCreator<SBMLDocument> plugin_AppearExtension(SbmlDocExtPoint, packageURIs);
    SBasePluginCreator<SBMLModel> plugin_ModelAppearExtension(ModelExtPoint, packageURIs);
    SBasePluginCreator<SBMLCompartment> plugin_CompartmentAppearExtension(CompExtPoint, packageURIs);
    appearExtension.addSBasePluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBasePluginCreator(&modelPluginCreator);
    appearExtension.addSBasePluginCreator(&compPluginCreator);

    int result = SBMLExtensionRegistry::getInstance().registerExtension(appearExtension);

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

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);

    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 = 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);
    SBaseExtensionPoint sbmlDocPluginExtPoint("core", SBML_DOCUMENT_PLUGIN);
    SBaseExtensionPoint modelPluginExtPoint("core", SBML_MODEL_PLUGIN);
    SBaseExtensionPoint compPluginExtPoint("core", SBML_COMPARTMENT_PLUGIN);
    SBaseExtensionPoint sbmlDocPluginCreatorExtPoint("core", SBML_DOCUMENT_PLUGIN_CREATOR);
    SBaseExtensionPoint modelPluginCreatorExtPoint("core", SBML_MODEL_PLUGIN_CREATOR);
    SBaseExtensionPoint compPluginCreatorExtPoint("core", SBML_COMPARTMENT_PLUGIN_CREATOR);

    int result = SBMLDocumentPluginCreator(sbmlDocExtPoint, packageURIs);
    SBMLExtensionRegistry::getInstance().addExtension(&appearExtension);
    appearExtension.addSBMLDocumentPluginCreator(&sbmlDocPluginCreator);
    appearExtension.addSBMLModelPluginCreator(&modelPluginCreator);
    appearExtension.addSBMLCompartmentPluginCreator(&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;
```

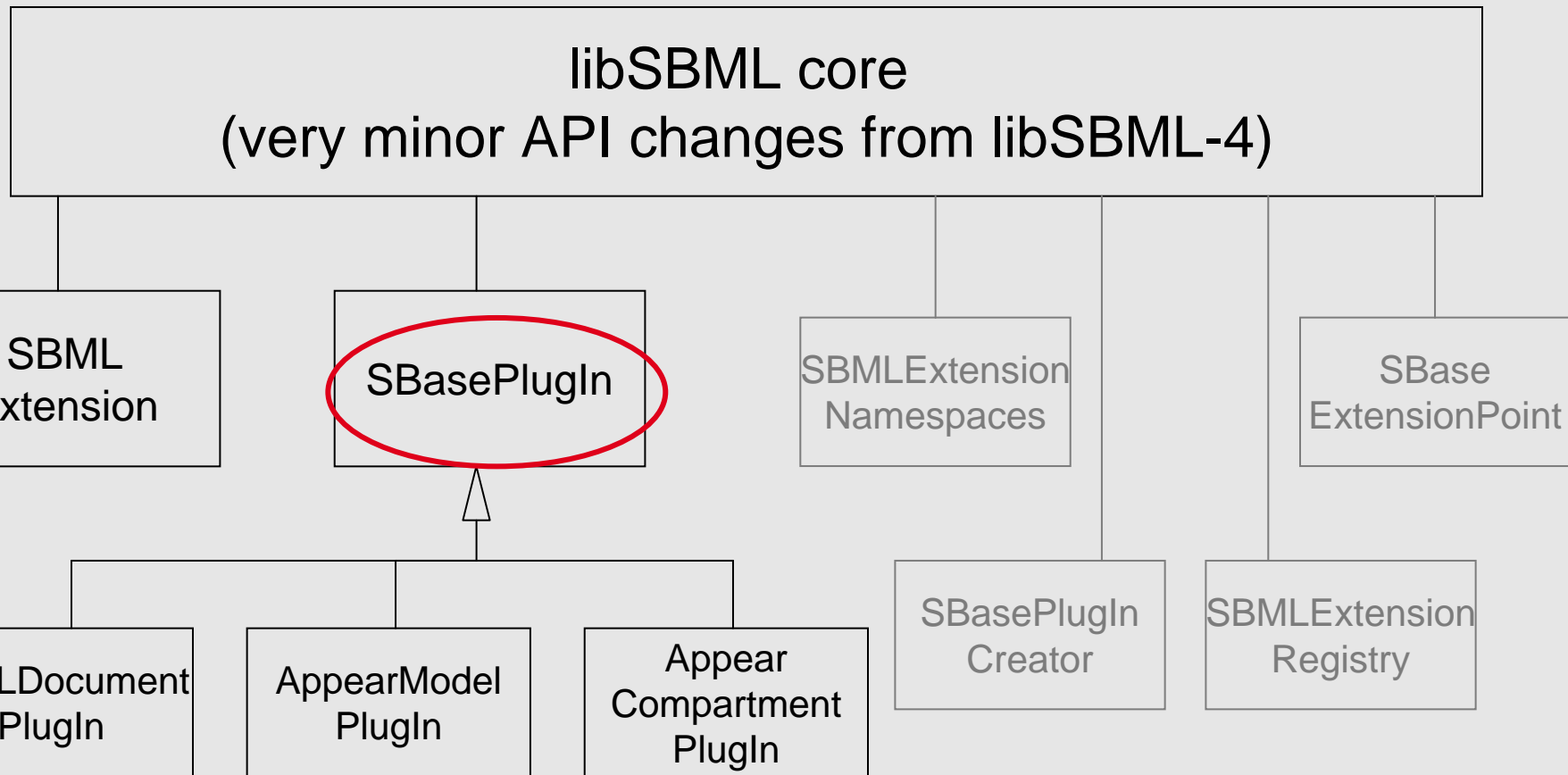
```
SBML_COMPARTMENT  
SBML_LAYOUT_CURVE  
SBML_GROUPS_MEMBER
```

# Implementing a package

## 2. Create the PlugIn classes

```
void  
AppearExtension::init()  
{  
    if (SBMLExtensionRegistry::getInstance().isRegistered(PackageName))  
    {  
        return;  
    }  
  
    SBasePluginCreator<SBMLDocumentPlugin, AppearExtension>  
    sbmlDocPluginCreator(sbmlDocExtPoint,packageURIs);  
  
    SBasePluginCreator<AppearModelPlugin, AppearExtension>  
    modelPluginCreator(modelExtPoint,packageURIs);  
  
    SBasePluginCreator<AppearCompartmentPlugin, AppearExtension>  
    compPluginCreator(compExtPoint,packageURIs);  
  
    int result = SBMLExtensionRegistry::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 void addExpectedAttributes(ExpectedAttributes& attributes);
    virtual ~AppearCompartmentPlugin ();
    AppearCompartmentPlugin& operator=(const
                                       AppearCompartmentPlugin& orig);
    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 (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

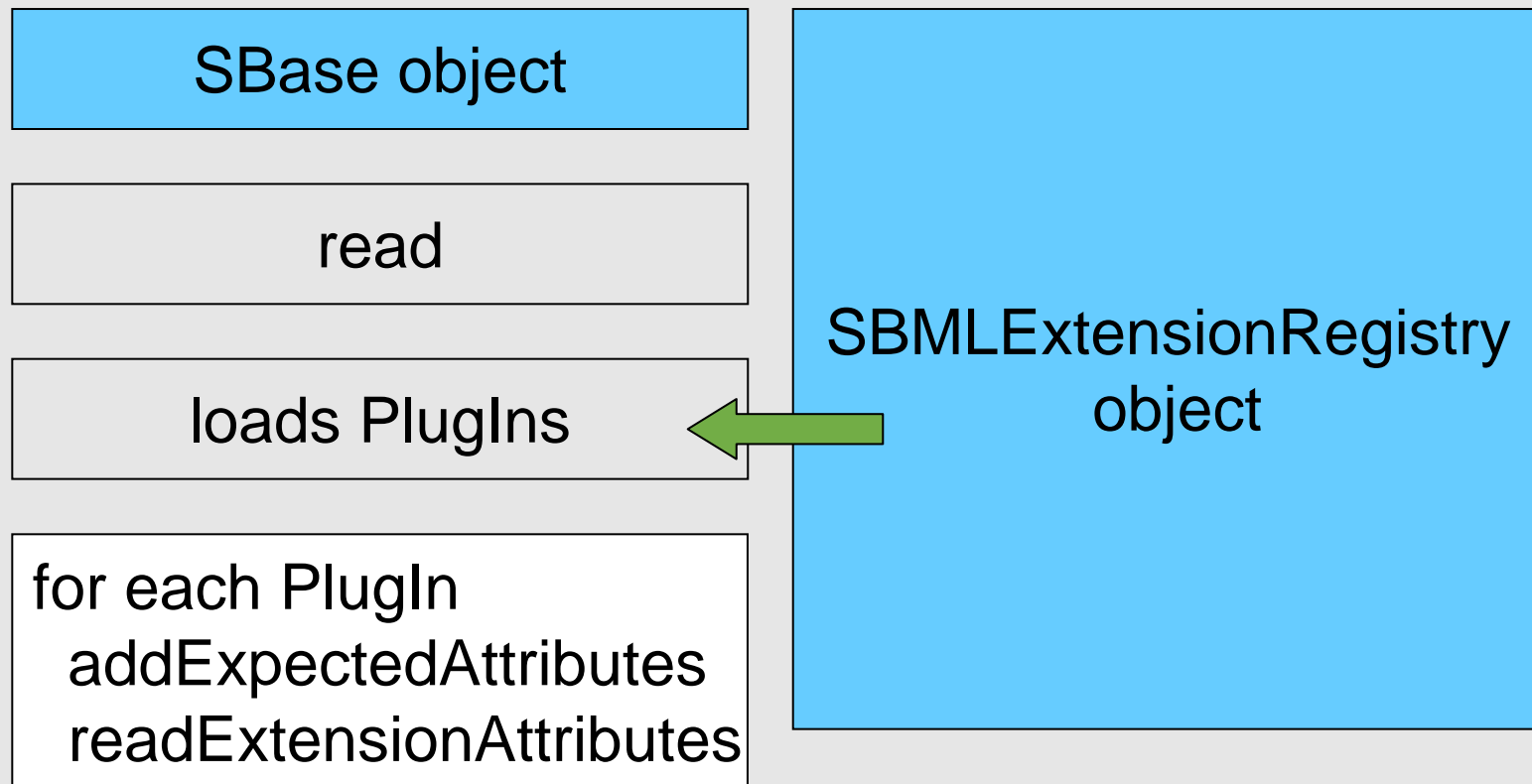
```
class AppearCompartmentPlugIn : public SBasePlugIn
{
public:
    AppearCompartmentPlugIn (const std::string &uri, const std::string &prefix, SBMLNamespaces *sbmlns);
    AppearCompartmentPlugIn(const AppearCompartmentPlugIn& orig);
    virtual ~AppearCompartmentPlugIn ();
    AppearCompartmentPlugIn& operator=(const AppearCompartmentPlugIn& orig);
    virtual AppearCompartmentPlugIn* clone () const;

    virtual void addExpectedAttributes(ExpectedAttributes& attributes);
    virtual void readAttributes (const XMLAttributes& attributes,
                                const ExpectedAttributes& expectedAttributes);
    virtual void writeAttributes (XMLOutputStream& stream) const;

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

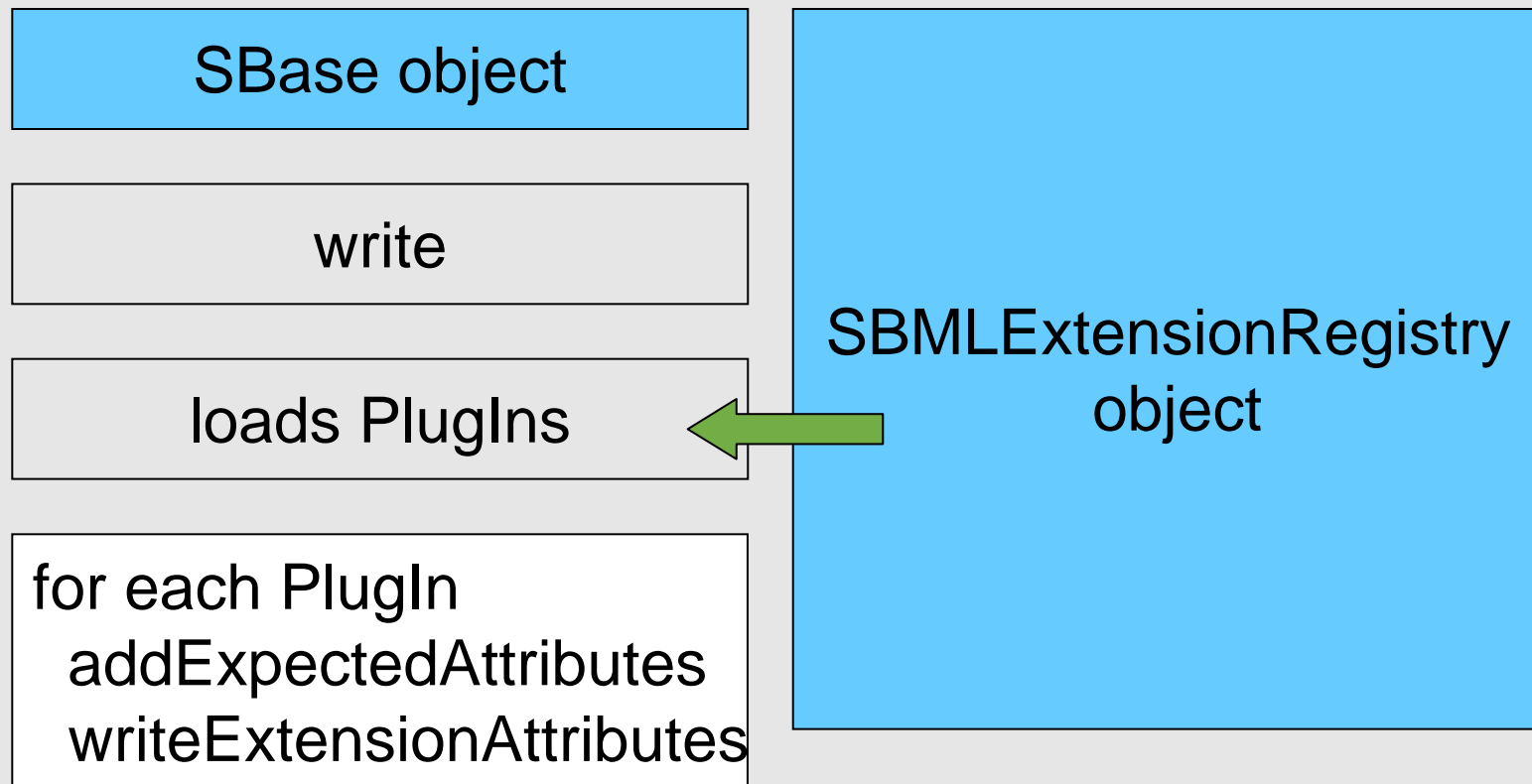
# Implementing a package

## 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,
                                const ExpectedAttributes& expectedAttributes);
    std::string getShape() const;
    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;
};
```

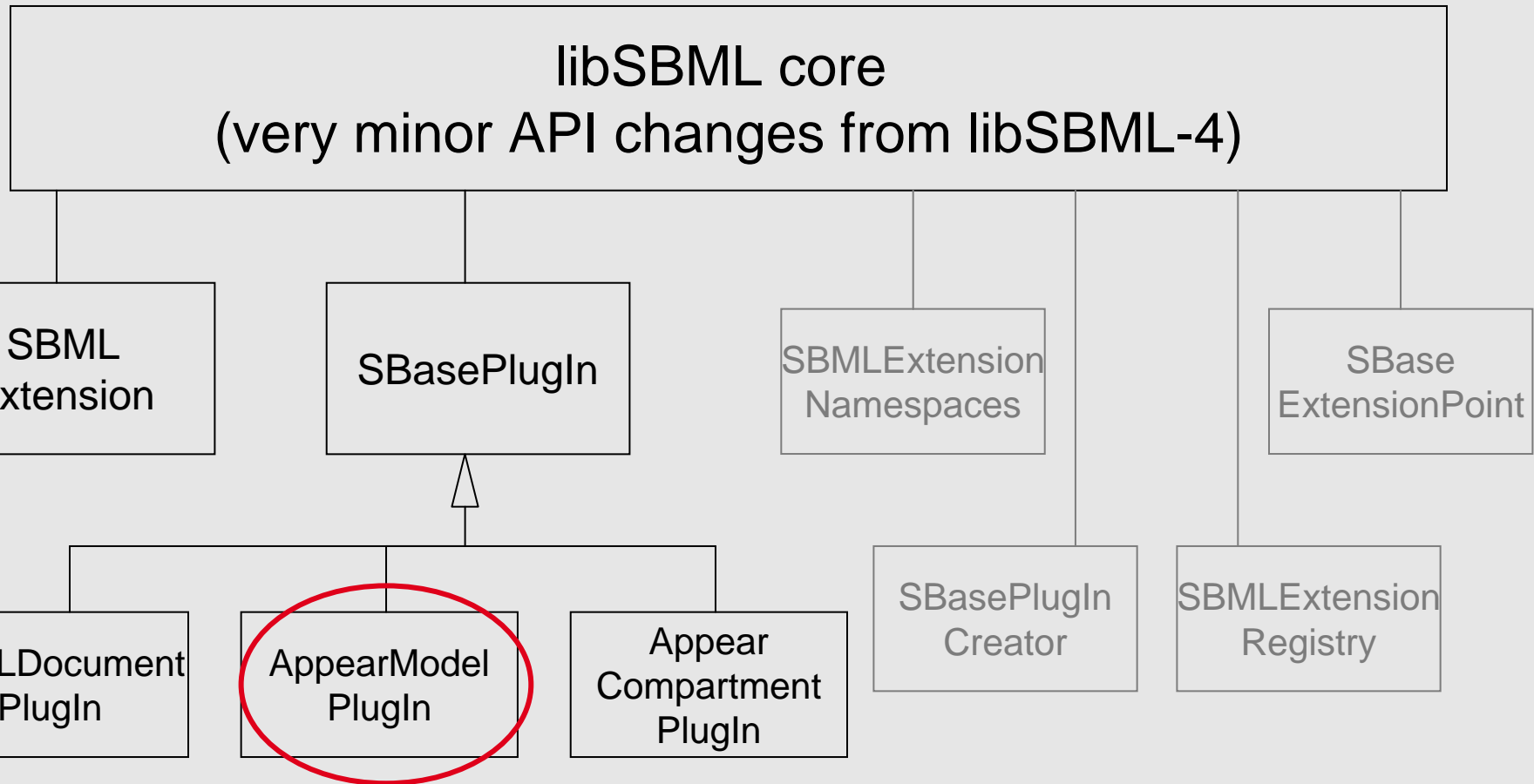
manipulate 'shape' attribute

# 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:
    Shape* 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;

    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 (XMLOutputStream& 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("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);
    document doc("sbml", 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 = m->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);

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

    Shape *s = m_plugin->createShape();
    s->setId("circle");
    AppearCompartmentPlugin * c_plugin = static_cast<AppearCompartmentPlugin*>
    (c->getPlugin("appear"));
    c_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"));

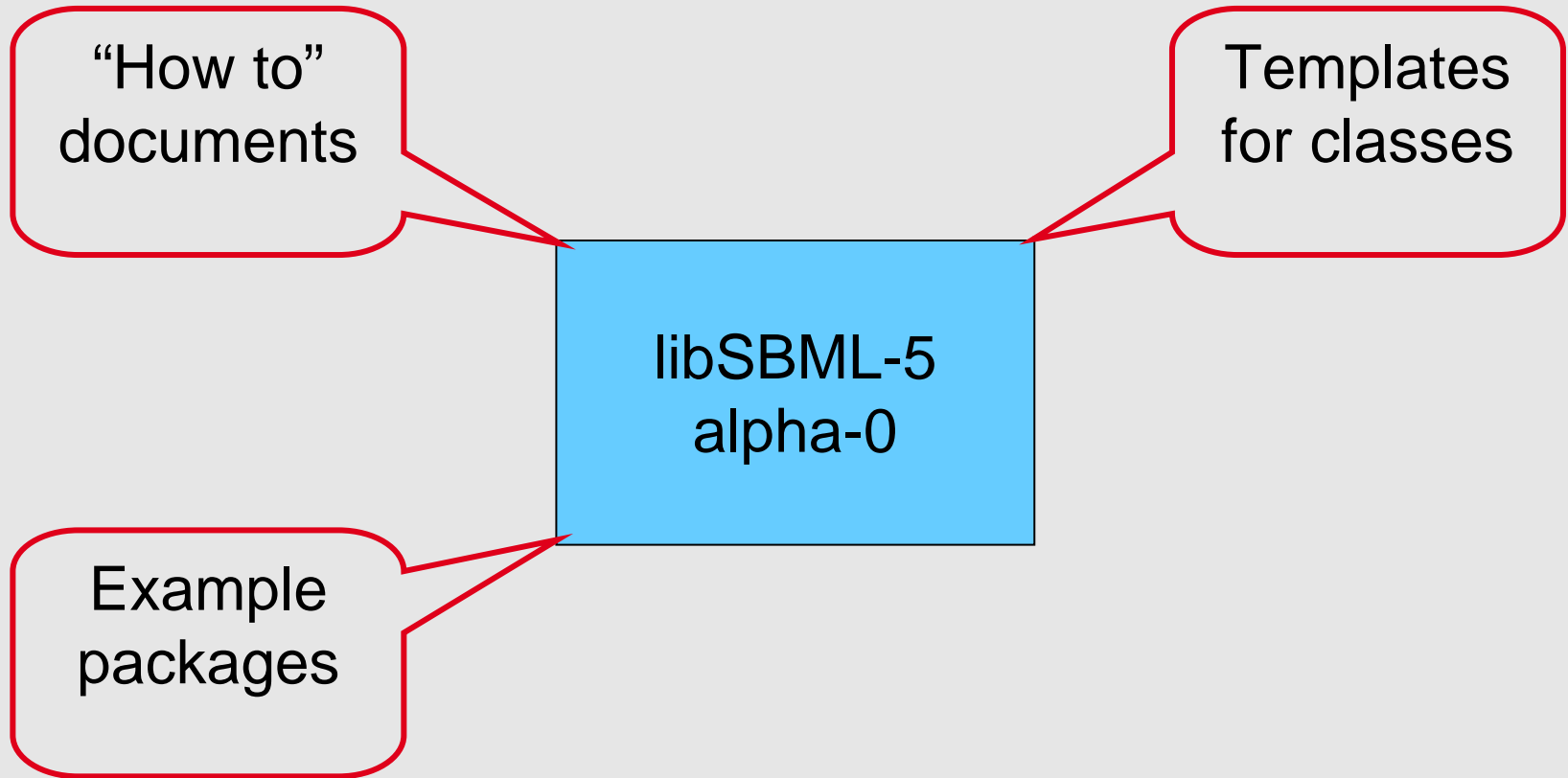
    Shape *s = m_plugin->createShape();
    s->setId("circle");
    AppearModelPlugin * c_plugin = static_cast<AppearModelPlugin*>.(c->getPlugin("appear"));
    plugin->setShape("circle");
    delete document;
}
```

# Implementing a package

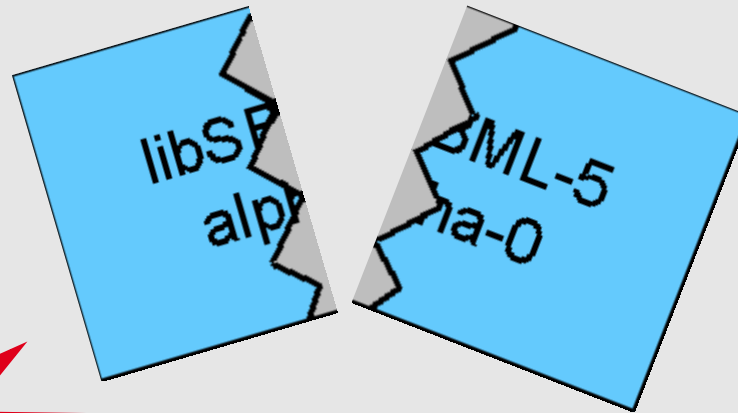
## 3. Using the package

```
<?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>
```

# libSBML-5



# libSBML-5



Please

# Acknowledgements



Akiya Jouraku  
Keio, Japan



Mike Hucka  
Caltech, USA