



BioGears 8.0 Roadmap

A. Baird¹, J. Carter¹, L. Marin¹, M. McDaniel¹, N.
Tatum¹, S. White¹

1. Applied Research Associates Inc.

Introduction

Purpose:

- To discuss planned upgrades for the BioGears API as we move towards a future 8.0 release
- To encourage community feedback on the impact of proposed changes
- Note that several features in this talk may be incorporated into minor releases of the 7.X line as we march towards the release of 8.0
- Any feature denoted in red is an API breaking change we believe must wait for a major version update. Priority of feature updates is there for leaning towards non API based modifications

BioGears 8.0 Development Roadmap

Feature Enhancements

- **Availability of Event Callback(Delegates, Signal/Slots)**
 - Lambda based event handling to simplify front in integration with physiology
- **FileIO fallback library for imbedded development**
 - New library for embedded development to reduce mandatory fileIO
- **Deprecate ScenarioExec**
 - Expose new Physiology Driver Framework
- **Scenario Language 2.0**
 - Support for logic control structures (Branch, Loop)
 - Action Interval fields (Reduce number of advance model time actions)
- **Language Hooks**
 - C Interface
 - C# and Python hooks

BioGears 8.0 Development Roadmap

API Changes

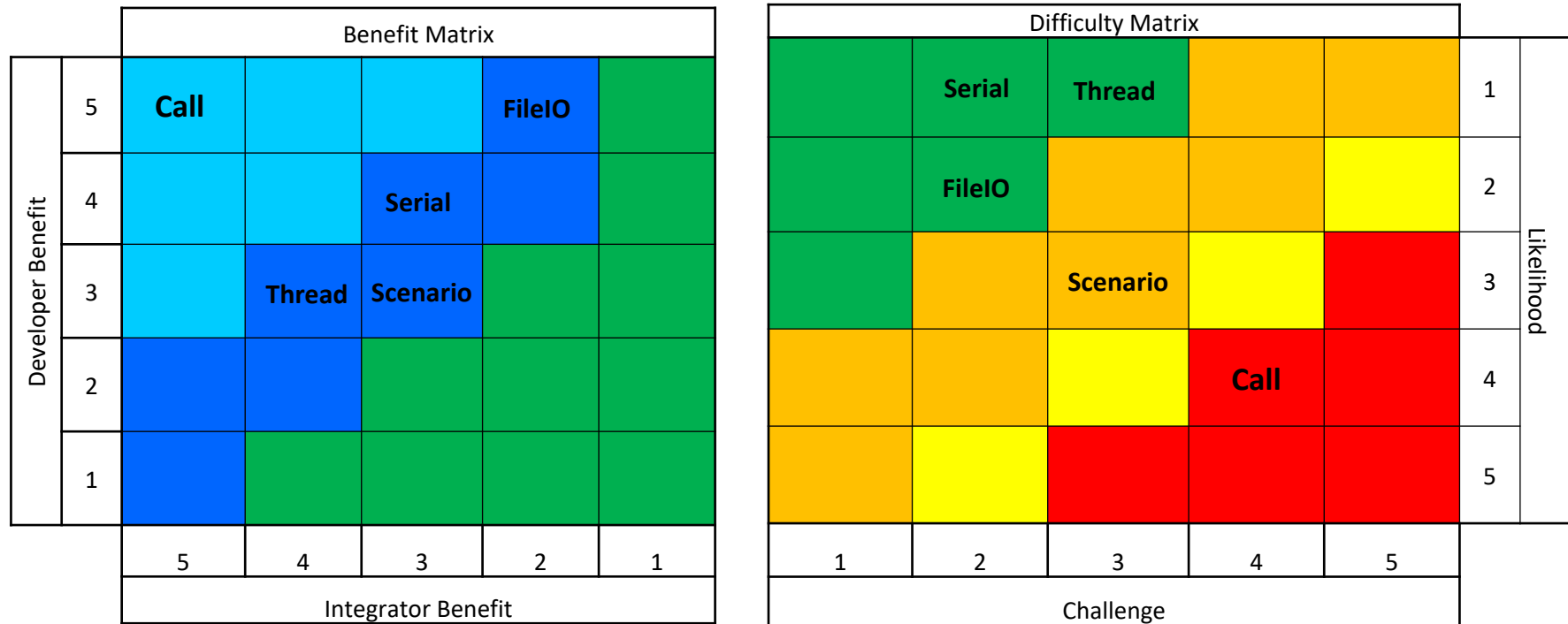
- **Merging of BioGears and BioGearsEngine**
 - Standardize naming of accessor functions for SESystem* returns and concreate system implementations.
 - Eliminate multi enharatance which is confusing in the API layer.
- **General API Changes / Modernization**
 - Return type changes
 - C++11 style changes
- **Serialization Overhaul**
 - Replacement of XML with JSON based format
 - Removal of serialization details from API layer

BioGears 8.0 Development Roadmap

Refactoring and quality of life changes.

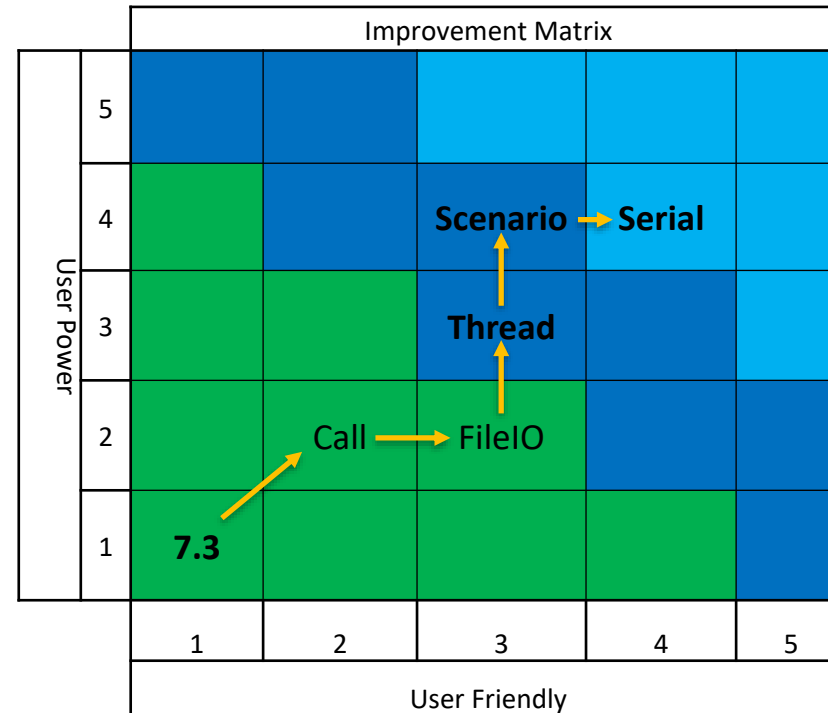
- **Audit of third-party dependencies**
 - Replaces Xerces-c and XSD Code Synthesis with libprotobuf
 - Replace log4cpp with internal logging code.
- **Scalar Unit Refactor and Replacement**
 - Expose arithmetic operators for SEScalar types
 - Migrate internal model code to arithmetic operators.
- **Uniform error/exception policy**
 - Catch all third party exceptions and expose them as BioGears Error type
 - Eliminate CDM exceptions in favor of Error model.
- **Uniform memory ownership model.**
 - Clarify when ownership is assumed by the library for passed objects
 - Expose copy and move constructors for all types to simplify semantics
- **Substance Definition Class**
 - New class for storing const substance definitions.
 - Eliminate need for Substance Manager to be passed to many classes.

Benefit vs Difficulty



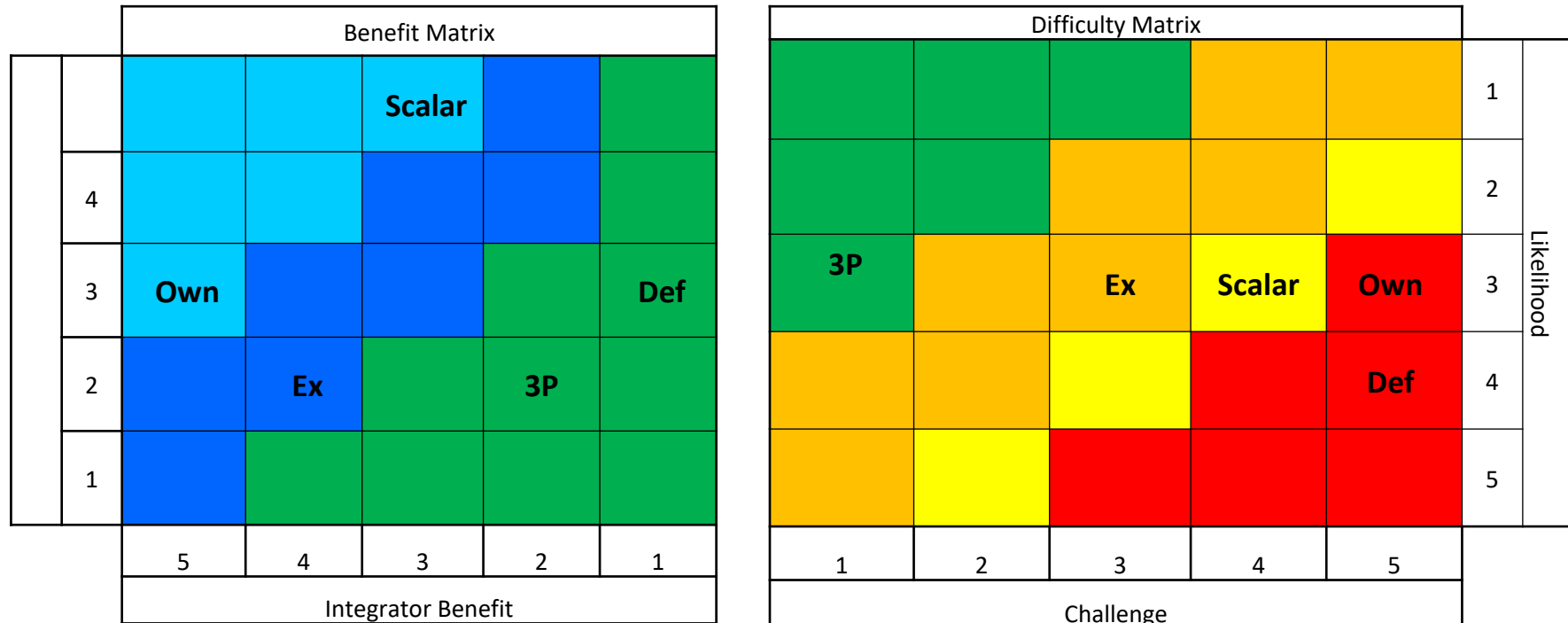
Key	Task
Call	Availability of Event Callback
FileIO	FileIO fallback library for imbedded development
Thread	Deprecate ScenarioExec
Scenario	Scenario Language 2.0
Serial	Serialization Overhaul

Overall 8.0 Benefits



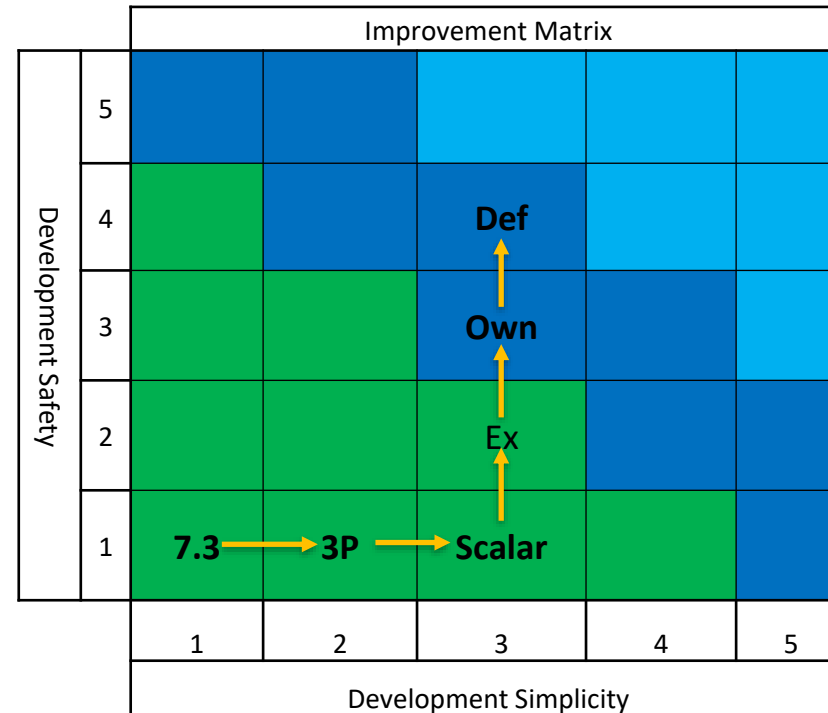
Key	Task	Benefits
Call	Availability of Event Callback	All Integrators
FileIO	FileIO fallback library for imbedded development	Embed Developers
Thread	Deprecate ScenarioExec	Front in projects
Scenario	Scenario Language 2.0	End Users
Serial	Serialization Overhaul	Windows Developers

Benefit vs Difficulty



Key	Goal
3p	Reduced third-party dependencies
Scalar	Scalar Unit Refactor and Replacement
Ex	Uniform error/exception policy
Own	Uniform memory ownership model
Def	Substance Definition Class

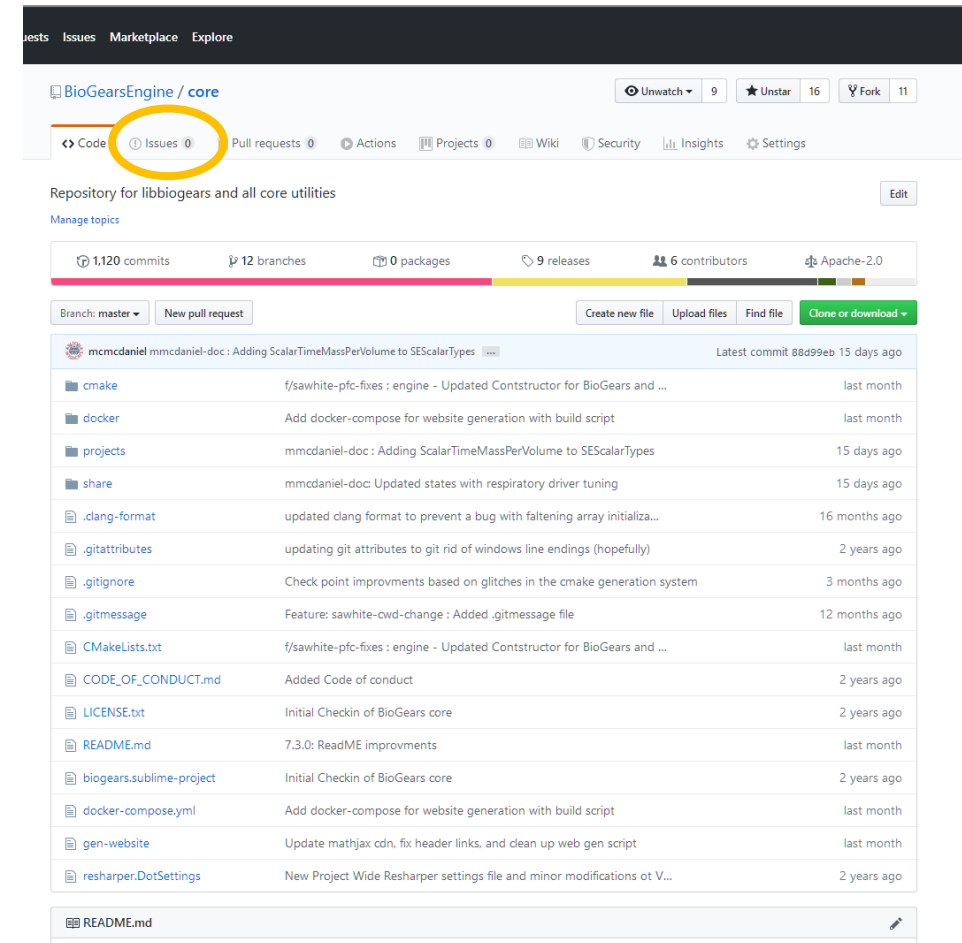
Overall 8.0 Benefits



Key	Task	Benefits
3p	Reduced third-party dependencies	All Integrators
Scalar	Scalar Unit Refactor and Replacement	Modeler
Ex	Uniform error/exception policy	All Integrators
Own	Uniform memory ownership model	All Integrators
Def	Substance Definition Class	Maintainers

Development Proposals

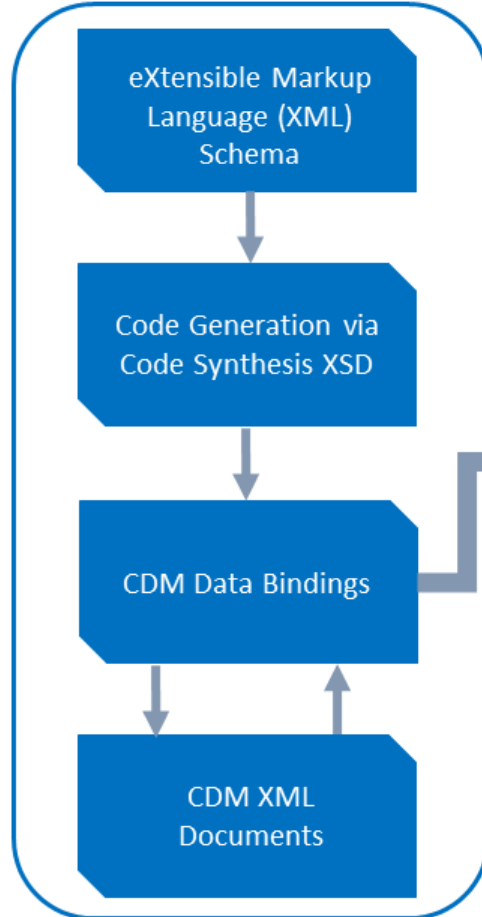
- BioGears wants your feedback.
 - Please create Issue tickets for suggested features.
- Process
 - Open an Issue with the enhancement label.
 - Work with development team to flush out the proposal and attach it to the ticket.
 - If capable work with development team to co-develop feature or provide pull request of proposed implementation



BioGears Architecture

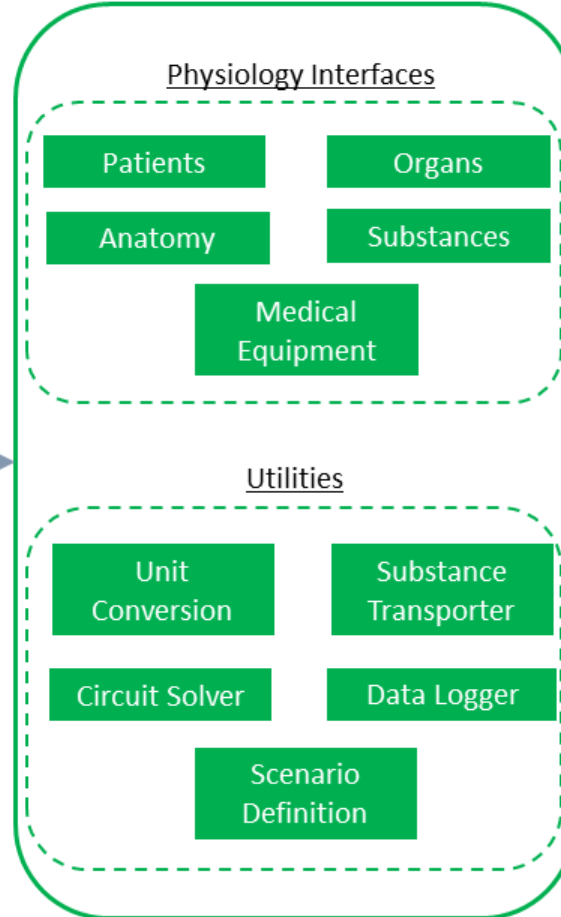
Common Data Model (CDM)

Communication standard for
physiology data transfer



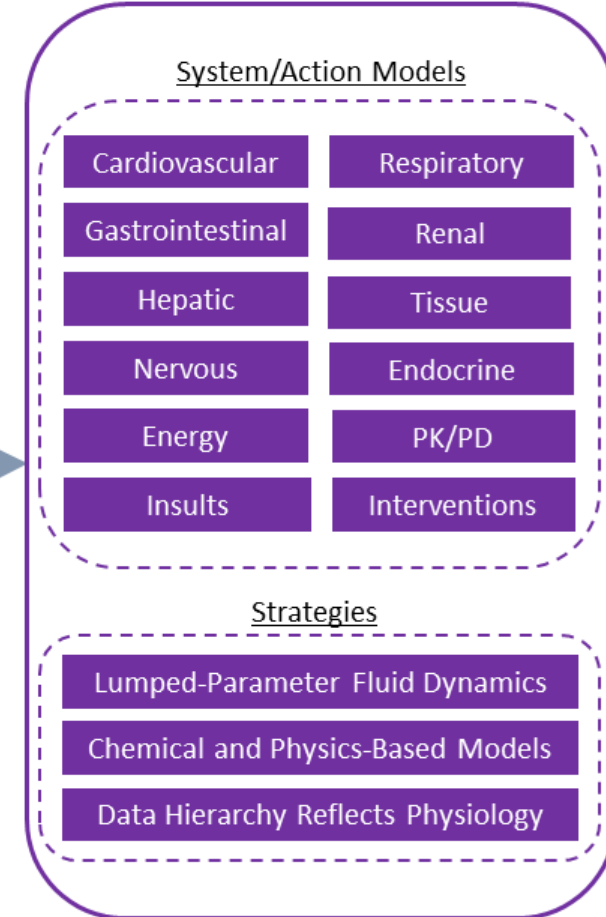
Synthetic Environment (SE)

C++ API that implements an abstract
Physiology Engine class



BioGears Engine

Physiology Engine instance
developed by ARA



Event Signal/Slots

Goals

- Ease the amount of code needed to detect common physiology events.
- Support functional programming paradigms for downstream integrators.
- Document best practices for reacting to physiology changes.

Who benefits

- C++ Integrators
- All GUI developers using BioGears
- Real-time BioGears embedded systems.

Signal/Slots Implementation

//Option 1: Separate Function Signatures

```
virtual void DeathFunc(std::function<void(void)> func) =0;
virtual void TachycardiaFunc(std::function<void(void)> func) =0;

engine->DeathFunc( deathSlot );
```

//Option 2: Connection Signature

```
enum class Signal {
    DEATH,
    TACHYCARDIA,
    ENTER_COMMA,
    EXIT_COMMA,
};

virtual void connect(Signal signal, std::function<void(void)> slot)=0;
```

//Both Option Use Cases are similar

```
auto deathSlot = [&]()
{
    isPatientDead = true;
    qWarn() << "Patient Has Died. Pausing Thread";
    bgThread.Pause(true);
};

engine->connect(Signal::OnDeath, deathSlot);
```

Implementation

- Work with modeling team to identify 2-5 events of interest
- Determine on Option 1 or Option 2 based API signatures
- Implement hooks and provide community howto's on new features
- Implement use in BioGears Visualizer as proof of concept.

Signal/Slots Rationale

Rationale for the changes

- API documentation makes responding to death surprisingly hard.
- Irreversible state can easily go unnoticed by integrators as no programmatic changes occur in BioGears.
- Most integrators need to write the same boilerplate bool checks for several BioGears Events
- In general log messages is the only way to determine if an event occurred in BioGears.
- Signal / Slots is a fairly reliable way to allow integrators to assign executable code directly in to our physiology models.

Take away

- Modern method of simulation event processing.
- Simplifies all GUI based BioGears integration.
- Should simplify hardware integration with BioGears

File IO Fallback

Goals

- Optional libbiogears_data library that contains all prerequisite IO.
- Removal of all assumed FILE IO to better support embedded development.
- Environment based controls for data roots.

Who benefits

- Embedded developers without traditional file system abstractions.
- Package maintainers trying to create clean integration.
- End users wanting to run BioGears from multiple directories.
- Mobile developers needing to deal with wall-gardened style file IO.

File IO Implementation

```
void generate_data_dir(std::path path);
void generate_schema_dir(std::path path);

void sha(eDataFile, std::path);
void sha(eSchema`File, std::path);
```

Implementation

- Modify BioGearsEngine to fall back to refer to ENV variables if
 - No runtime directory is not provide
 - If requested file is not resolved
- Implement optional new library libbiogears_data
 - Integrate file encoding using cmake with an existing utility (e.g *xxd*, *binfs*, or *objcopy*)
- All initial FileIO will fallback to libbiogears_data if
 - BIOGEARS_NO_FS was defined at compile time
 - Requested file is not found.
 - An IO Exception is thrown while reading the file.
- Enhance log messages to record when any fallback location was used. This will add transparency on if a local or static definition was used at runtime.

File IO Rationale

Rationale for the changes

- Eliminating file system assumptions simplifies development on mobile platforms.
- Increases support for Medical trainer integration.
- Supporting for reading ENV variables will allow for the executable to be separated from its data directory.
- Allows libbiogears to recreate its own runtime directory structure when a corruption is detected.

Take away

- Improved cross-platform support.
- Robust data directory protections
- Easier to maintain for frontend integrators
- Major request of embedded development teams

New Environment Variables

Variable	Purpose
BIOGEARS_SERIAL_DIR	CDM Schema definition location
BIOGEARS_DATA_DIR	Location of Patient, Substance, and Environment definition files.

Scenario Language Enhancements

Goals

- Context free scenario scripting
 - If, Then, Else
 - Range Loops
 - (Stretch Goals) Data Request Conditionals
- Integrate application interval to Action declaration
- Ability to perturb action parameters based on given distribution [normal, uniform, exponential].
- Simplified batch runs for generating population results.

Who benefits

- Non C++ users who want to use BioGears as a utility
- Researchers looking to create population results.
- Utility developers using our scenario driver framework.

Scenario Language Implementation

Name	Pseudo code	Purpose
BRANCH AND FORK	BRANCH(X)	Create a parallel scenarios one where X was enacted and one that wasn't
BRANCH AND FORK	BRANCH(X,Y)	Create parallel scenario one where X occurred and one where Y occurred
LOOP X TIMES	LOOP(5;<ActionDataList>)	Assign a new variable to a variable

Implementation

- Expand CDM definitions for actions to include duration
- Create CDM IF and LOOP types
- Expand ScenarioExec to support new constructs.
- Create CDM type for statistical distribution
- Create CDM type for perturb
- Expand Scenario Exec to support input perturbation
- Expand Sceario Exec to support data request based conditions

Name	Pseudo code	Purpose
var declaration	NEW(X);	Declare a variable to store for later usage.
Assignment	SET(X,2); DATAREQUEST(A,"HeartRate)	Assign a new variable to a variable
Increment	ADD(X,1)	Increase the value of a variable by a given amount
decrement	DEC(X,3)	Decrease the value of a variable by a given amount
Equality	EQUAL(X,Y)	Ability to determine if two values are equivalent,
Compare	GREATER(X,Y)	Compare if one value is greater then the other
Conditional Branch	IF(EQUAL(X,Y),ADD(X,1),DEC(X,1))	Apply Action 1 if Condition is true else Apply Action 2
While Loop	WHILE(GREATER(Y,X);ADD(X,1))	Apply Action until Condition is no longer true
Boolean Logic	AND,OR	Helper functions for determining if two conditions are true. Reduces complexity of branch statments
Negation	NOT(GREATER(X,Y))	Return the inverse of a Boolean operation. While not required its convenience is undeniable.

Scenario Language Rationale

Rationale for the changes

- Machine learning projects often need to generate large amounts of population data.
- Simplifying the encoding of scenario variability will greatly reduce the number of files a project needs to track.
- BioGear's time skipping requires the generating of large state trees.
- Lack of time encoding within an action complicates implementation.

```
<Action xsi:type="PainStimulusData">
<Comment>Puncture wound in right hand.</Comment>
<duration value="1" unit="hour"/>
<severity value=".5" />
<compartment value="RightHand"/>
</Action>
```

Take away

- Reduced number of State files most projects need to track.
- Simple way of encoding state creation graphs.
- Makes the ability of generating population variation data possible.

```
<Action xsi:type="SubstanceInfusionData">
<Comment>Morphine Drip.</Comment>
<duration value="15" unit="minutes"/>
<Substance value="Morphine" />
<concentration value="10" unit="mL/L" />
<volume value="1" unit="L"/>
</Action>
```

Serialization

Goals

- Removal of Serialization details from API Layer.
- Replacement of XML implementation with JSON format.

Who benefits

- Improved serialization will give performance benefits to all end users.
- Smaller serialized states will benefit embedded developers.
- Windows integrators benefit from the removal of exported STL symbols.

Serialization Implementation

```
[[deprecated]]
virtual CDM::ScalarData* UnLoad() const;

[[deprecated]]
virtual void Load(const CDM::ScalarData& in);

virtual void marshall(const std::istream& is);
virtual void unmarshall(const std::ostream& os) const;

virtual std::istream& operator<< (const std::istream& is);
virtual std::ostream& operator>>(const std::ostream& os) const;

namespace biogears {
    //Implementation pending testing may prefer overloading
    template<typename T>
    std::ostream& operator<<(std::ostream& os, const T& obj)
    {
        obj.unmarshall(os);
        return os;
    }
    template<typename T>
    std::istream& operator>>(std::istream& is, const T& obj)
    {
        obj.marshall(is);
        return is;
    }
}
```

Implementation

- All SE Classes and engine classes now friends with associated serialize functions
- Load/UnLoad duplicated in biogears::cdm::serializer and biogears::engine::serializer.
- Load/UnLoad in SE and derived classes removed from API
- Addition of Copy and Move constructors for all BioGears types

Serialization Rationale

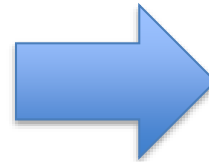
Rationale for the changes

- XSD's template code exports std::basic_string when used with MSVC making linker troubles for all Windows Integrators
- Serialization exposed to our API locks us in to a single solution and complicates binary compatibility
- Protobuf will allow binary and human readable serialization
- Protobuf has a better license fit with BioGears

Take away

- Faster Integration
- Increased performance
- Easier to maintain

```
<Action xsi:type="EnvironmentChangeData">
  <Comment>Insult: Increased temperature </Comment>
  <Conditions>
    <Name>LocalEnvironment</Name>
    <AmbientTemperature value="20.0" unit="degC"/>
    <MeanRadiantTemperature value="20.0" unit="degC"/>
    <RespirationAmbientTemperature value="20.0" unit="degC"/>
  </Conditions>
</Action>
```












```
EnvironmentChangeData : {
  Comment : "Insult: Increased Temperature"
  ,Conditions: { Name: "LocalEnvironment" }
  , AmbientTemperature : {
    value: 20.0 ,unit: "degree-celcius"
  }
  , MeanRadiantTemperature : {
    value: 20.0 ,unit: "degree-celcius"
  }
  , RespirationAmbientTemperature : {
    value: 20.0 ,unit: "degree-celcius"
  }
}
```


Acknowledgements

BioGears is aided by the works of many open source projects and organizations.

Projects

-  Xerces
-  XSD Code Synthesis
-  Boost
-  Log4Cpp
-  Python
-  Python Buildbot
-  Visual Studio Code
-  Ubuntu
-  Google Test

Organizations

- JPC-1
- Applied Research Associates 
- University of Washington

References

1. <http://biogears.dev>
2. <http://biogears-engine.dev>
3. <http://boost.org>
4. <https://developers.google.com/protocol-buffers>
5. <http://github.com/biogearsengine>

Language Hooks

Goals

- Provide language hooks for the most common integration platforms.
- Simple language hooks which can
 - Create a physiology engine
 - Advance time
 - Retrieve data request

Rationale

- One of our most requested features with conference attendees.
- Simplifies the barrier for developing BioGears
- Unity is C# based and the most popular rendering engine in the world
- Python is one of the most popular scripting languages with several libraries that could be used to rapidly integrate with BioGears.

Suggested Languages

- C {Common Core for language hooks}
- Python 3.X {Prototyping}
- C# {Unity}

Who benefits

- New Integrators
- Next Generation of Developers
- Community through outreach

Reminder of Planned API Changes

Pending API Changes

- Merging of BioGears and BioGears Engine Classes
 - Simplify BioGears use of const and reduce complexity of gaining access to concrete features.
 - Casting to Physiology Engine will behave the same as existing 7.4 API.
- Physiology Engine – Uniform naming of assessors to SESystem*
 - Clarity of expected return type Renal vs SERenalSystem will reduce the rampup time of new integrators.
- Physiology Engine – Preference over SESystem& and SESystem* to communicate ownership
 - Reference returns will communicate lifetime of item is the same as the producer
 - PTR returns will communicate management of lifetime by caller
 - New release functions will allow manual cleanup of PTR returns.
- const correctness in Physiology Engine
 - Allow all data request from a const Physiology Engine
 - Allow AdvanceModelTime of a const Physiology Engine
 - Prevent all Set calls of a const Physiology Engine
 - Eliminate const/non const return type variants for API clarity if possible
- Removal of Load/Unload functions from code

Reminder of API Changes 2

- Renaming of `create_*` to `make` or `make_*` to align with C++11
 - More intuitive API based on modern coding standards
- Exposing protected constructors
 - Allows use of `make_unique` and `make_shared` std calls
 - Little reason to prevent construction of types
 - Allows future dependency injection to simplify unit test
- Removal of deprecated calls
 - Moving forward we will remove deprecated calls roughly one year after they have been marked unless significant community pushback is recorded.

Dependency Review

Goals

- Simplify building of BioGears.
- Ensure all dependencies maintained.

Who benefits

- Any user who builds BioGears from source
- Non traditional embedded platform developers
- Core development team

Dependency Review Implementation

Implementation

- Survey of all possible logging implantation alternatives
- Public test branches for any alternative consideration
- When possible CMake will pull new dependencies automatically if they are missing.

Dependency Review Rationale

Rationale for the changes

- Log4cpp is no longer maintained. May contain exploitive behavior or be the source of IO instability
- Logging code is very verbose for a single threaded program
- XSD exporting std::string is a burden on Windows Developers
- XSD license could conflict with some use cases of BioGears.
- Alternative logging code may simplify use on Android and MacOS platforms.

Take away

- Simplified Building Process.
- Will not needlessly expand third party dependencies.
- Only Apache 2.0 compatible license models will be chosen as acceptable replacements.

Scalar Refactor

Goals

- Arithmetic Operators for all SEScalar Types
- Reduced use of SetValue and GetValue functions
- Compile time conversions to enhance BioGears performance
- Removal of string comparisons to determine type conversion

Who benefits

- BioGears Modelers
- Embedded Integration Projects
- API users working with DataRequest

Scalar Refactor Implementation

Arithmetic operators will catch modeler errors earlier by generating compile time error messages

Example:

No operator takes a left and SEScalarPower and a right hand SEScalarLength*

```
SEScalarArea operator*(SEScalarLength const&, SEScalarLength const&);
SEScalarVolume operator*(SEScalarArea const&, SEScalarLength const&);
SEScalarVolume operator*(SEScalarLength const&, SEScalarArea const&);

SEScalarVolumePerTime operator/(SEScalarVolume const&, SEScalarTime
const&);

SEScalarPower operator+(SEScalarPower const&, SEScalarPower const&);
```

Implementation

- Implement SEScalar and SEScalarQuantity using Types.h or an equivalent compile time dimensional analysis header over current conversion engine
- Define arithmetic operations for common combinations.
 - Addition
 - Subtraction
 - Multiplication
 - Division
- Publish public test branches with benchmark harness for community review
- Migrate model code to new type eliminating type mismatch common from manual value retrieval

We will ensure compatibility with existing DataTrack and Logging, but will likely simplify the SEScalarUnit interface.

```
class SEScalarUnit {
    virtual CCompoundUnit GetUnit() const = 0;
    virtual double GetValue() const = 0;
}

class CCompoundUnit {
    virtual std::string ToString() const = 0;
```

Scalar Refactor Rationale

Rationale for the changes

- Current method of Get/Set Value for type conversion does not allow for computer assisted type match verification
- Using operator overloading will provide modelers with clear compile time errors for type mismatch
- Using multiplication and division operators allows for type-safe assignments in model code.
- Removing string comparison on get value calls should provide a small but noticeable performance upgrade (0-5%)

Take away

- Faster Model Code
- Faster Model Development
- Reduced manual code review as the compiler will now be able to verify all units match in model code.

Uniform Memory Model

Goals

- Refactor API with uniform use of T*, T&, and T
 - Consistent use of const to denote mutability of passed or returned parameter.
 - Assume ownership of received T*
 - Maintain ownership of T* if returned and expose release function if data can be deleted early.
 - Document assumption of lifetime for all passed T& and T const &
- Document side-effects and deviations
- Copy & Move constructors for all types
- Migrate members of SEScalar* to SEScalar where possible
- Migrate T& members to T*
- Use std::shared_ptr and std::weak_ptr for shared objects

Who benefits

- BioGears Modelers
- Embedded Integration Projects
- API users working with DataRequest

Uniform Memory Implementation

- Largest and possibly most disruptive change in the 8.0 Roadmap.
- Very hard to fix in a transparent way.
- BioGears will implement through a cycle of document, annotate, modify as we work through the code base.
- Starting with SEAction and moving through SESystem derived classes.

Implementation

- Annotate all member functions with pending changes
- Provide migration guide as changes are implemented
- Implement Copy/Move constructors for all Marshall/UnMarshall clones.
- When possible provide new signature as parallel call one release prior to deprecation
- Migrate members in waves to limit disruption.

Uniform Memory Model Rationale

Rationale for the changes

- Inconsistent ownership across calls is the source of many integration bugs.
- UnLoad functions are memory leaks on windows platforms.
- Variation of internal members as references instead of ptrs make copy and move constructors hard to write.

Take away

- BioGears has several unusual bugs due to this lack of transparency.
- To much variation to properly document.
- All integrators and core team would benefit from API uniformity.

Function	Ownership Behavior
BioGearsEngine(Logger*)	Does not assume ownership; No Clone
ProcessAction(SEAction*)	Does not assume ownership; Preforms Deep Clone
SEScenario(SESubstanceManager*)	Does not assume ownership; No Clone; Lifetime of SESubstanceManager must exceed that of SEScenario
CDM::TypeData* UnLoad()	Allocates CDM::TypeData assumes caller will manage memory. Causes memory leaks across DLL bounds.
SEScalar* GetMember()	Allocates Member if it does not exist; Does not assume caller will manager memory; BioGears will crash if caller deletes PTR.
PhysiologyEngineDynamicStabilization::AddConditionCriteria(PhysiologyEngineDynamicStabilizationCriteria& criteria)	Stores PTR of Reference in Vector; Assumes ownership of memory passed; Very unusual C++ behavior.

Uniform Error Handling

Goals

- Biogears::Error based exception handling for all biogears calls
- Internal Exception handling for all IO errors.
- Robust behavior for AdvanceModelTime when a previous exception has occurred.
- Clear documentation on how integrators can avoid program crashes when a BioGears based fault occurs.

Who benefits

- All Integrators
- BioGears Core Team
- BioGears Utility Users

Uniform Error Rationale

Rationale for the changes

- BioGears 7.x throws exceptions across DLL bounds which is considered bad practice.
- BioGears 7.x throws transitive exceptions which are undocumented.
- It is almost impossible to use BioGears on a runtime with out exception support.

Take away

- Less Crashes
- Better Documentation
- Improved platform support
- Cleaner error handling

Uniform Error Handling

- Mostly affects serialization routines and logging.
- Could be used to improve user feedback for AdvancedModel Time.
- BioGears Is experimenting for an exceptionless error model.

Implementation

- Audit all IO calls
- Audit all occurrences of throw.
- Add doxygen markers for all functions which throw and include exact expectation type and conditions for the throw.
- Catch all third party and std exception and bubble up BioGears specific error codes.

Substance Definition Class

Goals

- Separate Substance Definitions from Substance Management
- Eliminate constructor arguments to SubstanceManager for DataTrack and SEScenario

Who benefits

- All Integrators
- UI Developers

Substance Definition Rationale

Rationale for the changes

- Substance directory contents almost never change during execution.
- SESubstanceManager creates circular lifetime deps for many high level classes.
- This is the only way to remove the entanglement.

Take away

- SESubstanceManagers lifetime causes development issues
- SESubstanceDefinitions would simplify model construction