University of Louisville
ThinkIR: The University of Louisville's Institutional Repository

Electronic Theses and Dissertations

12-2021

Update of MultiCellular data standard to match PhysiCell "compact" output.

Reid A Honeycutt
University of Louisville

Follow this and additional works at: https://ir.library.louisville.edu/etd

 Part of the Other Biomedical Engineering and Bioengineering Commons

Recommended Citation

Honeycutt, Reid A, "Update of MultiCellular data standard to match PhysiCell "compact" output." (2021). *Electronic Theses and Dissertations.* Paper 3925.
https://doi.org/10.18297/etd/3925

This Master's Thesis is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.

UPDATE OF MULTICELLULAR DATA STANDARD TO MATCH PHYSICELL "COMPACT" OUTPUT

By

Reid Alan Honeycutt
B.S., University of Louisville, 2020

A Thesis
Submitted to the Faculty of the
University of Louisville
J.B. Speed School of Engineering
as Partial Fulfillment of the Requirements
for the Professional Degree

MASTER OF ENGINEERING

Department of Bioengineering

December 2021

UPDATE OF MULTICELLULAR DATA STANDARD TO MATCH PHYSICELL "COMPACT" OUTPUT

Submitted by: 
Reid Alan Honeycutt

A Thesis Approved On

December 2nd, 2021

(Date)

By the Following Reading and Examination Committee:


Hermann B. Frieboes, Thesis Director


Dr. Nihat Altiparmak


Dr. Joseph Chen

ACKNOWLEDGEMENTS

I owe many people for the support they have offered during my completion of this project. This thesis project would have been impossible without their cooperation and guidance.

I would like to thank my thesis director, Dr. Hermann Frieboes, for his guidance and unfailing humor. Undertaking this project under his tutelage has deepened my understanding of bioengineering and the ways it can impact other fields. I owe him for the opportunity to work on this project and to expand my knowledge and skills.

I would also like to thank Dr. Samuel Friedman for lending me his expertise and for bearing with my endless questions. I would have been seriously lost and confused had I not been able to rely on his experience.

Finally, I would like to thank my family and friends for their continual support and encouragement throughout this arduous process.

ABSTRACT

In the field of multicellular biology, it is currently very difficult for researchers to share digital data, due to a lack of standards for said data. Each laboratory will collect data from a particular project, which will describe only the features relevant to their research, and then present that data using written descriptions, charts, graphs, and images. This makes it difficult for the data produced by these studies be leveraged in other research efforts by scientists studying similar phenomena. An open-source, universal standard that attempts to meet this need is the MultiCellular Data Standard (MultiCellDS). MultiCellDS is described as a "a community-developed standard to functionally describe cell phenotypes with contextual information from the microenvironment". A program called PhysiCell utilizes a modified, compact version of the MultiCellularData Standard through use of "custom" data tags that bypass validation. The goal of this project is to update MultiCellDS to reflect the data format utilized by PhysiCell so that PhysiCell files can be successfully validated by the standard. Additionally, a script is required to update PhysiCell files to remove the "custom" data tags, which are discouraged in the standard.

TABLE OF CONTENTS

Page

APPROVAL PAGE ii

ACKNOWLEDGEMENTS iii

ABSTRACT iv

TABLE OF CONTENTS v

LIST OF TABLES vi

LIST OF FIGURES vii

I. INTRODUCTION 1

A. Background 1

II. PROCEDURE 7

A. Project Overview 7

III. RESULTS AND DISCUSSION OF RESULTS 14

A. Validation Testing 14

IV. CONCLUSIONS 17

V. RECOMMENDATIONS 18

REFERENCES CITED 20

APPENDIX I. 21

APPENDIX II. 23

APPENDIX III. 24

VITA 26

LIST OF TABLES

Page

Table 1 - Validation Testing Results. 16

LIST OF FIGURES

Page

Figure 1 - Central Repository for Storing Clinical, Experimental, and Modelling Insights (Friedman, Anderson et al. 2016) 2

Figure 2 - MultiCellDS Glossary of Terms (Friedman, Anderson et al. 2016) 4

Figure 3 - A PhysiCell File With A “custom” Element, Which Contains Children Not Allowed by The Standard 5

Figure 4 - Flowchart Showing Focus of Project 7

Figure 5 - Flowchart Showing How Standard Was Updated 8

Figure 6 - Flowchart Showing How Update Script Was Written 8

Figure 7 - Example of The Hierarchical nature of XML Schema. 10

Figure 8 – Example validation of a PhysiCell File. The red box shows where the “custom” data tag used to be. 11

Figure 9 - PhysiCell File Before Update script 13

Figure 10 - PhysiCell file after update script. 13

Figure 11 - Validation of an "uncustomed" version 15

Figure 12 - Validation of Updated Version 15

I. INTRODUCTION

A. Background

In the field of multicellular biology, it is currently very difficult for researchers to share digital data, due to a lack of standards for said data. Each laboratory will collect data from a particular project, which will describe only the features relevant to their research, and then present that data using written descriptions, charts, graphs, and images. This makes it difficult for the data produced by these studies be leveraged in other research efforts by scientists studying similar phenomena. In addition, it means that teams are likely spending time to create software-based tools to create and analyze data that are useless outside of their own work. This phenomenon, known as data siloing, greatly reduces the speed at which research can be done because it makes it difficult for studies on similar topics to build off of each other. This issue could be rectified by a universally adopted data standard for describing multicellular systems. Such a standard would allow data to be shared effortlessly and enable the creation of central repositories to store both clinical and experimental data.

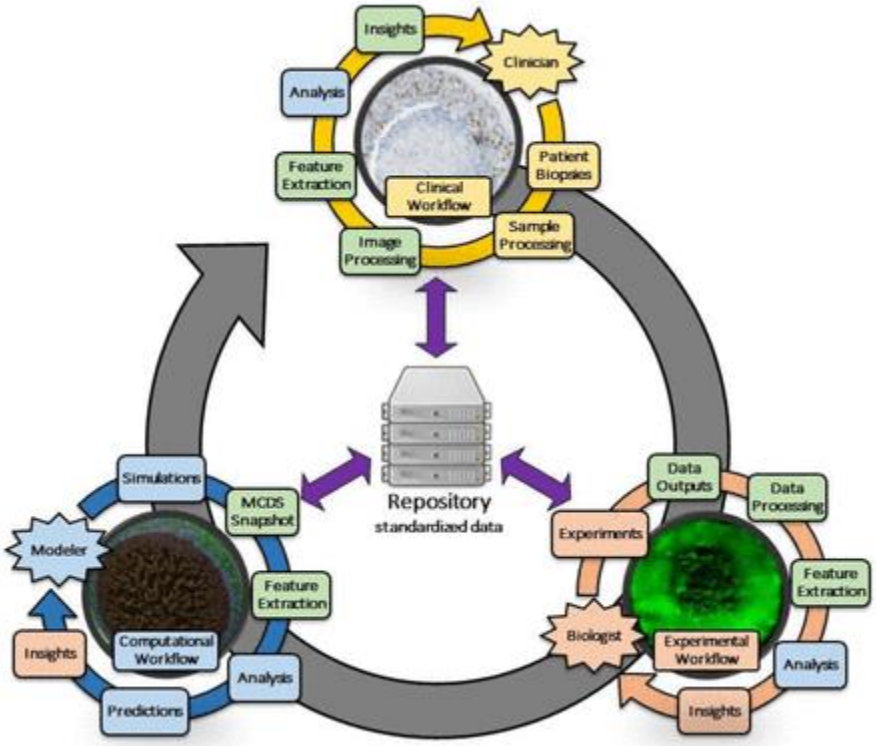

Figure 1 - Central Repository for Storing Clinical, Experimental, and Modelling Insights (Friedman, Anderson et al. 2016)

It would also allow for the creation of open-source software for collecting, analyzing, and formatting data that could be used to study any multicellular system. These tools would not only increase the ease with which new studies of multicellular systems could be conducted, but their interoperability would also make studies that utilize them more reproducible.

Standardizing the way that multicellular biological data is formatted and stored would also simplify software-driven efforts to model and simulate multicellular systems. Technologies for assessing single-cell systems are advancing rapidly. High-throughput multi-omics assays, along with techniques for manipulating and tracking cell histories mean that we have more information than ever about single cell systems. As these technologies begin to be applied to multicellular systems, large, high-resolution datasets

describing the chemical and behavioral characteristics of many cells within a multicellular system (Macklin 2019). Incredibly descriptive data such as these would enable modelers to simulate the behavior of multicellular systems. By continuously improving models to the point where they can accurately simulate real cellular systems, researchers would be able to run “trial” experiments computationally. This would save time and resources and allow investigators to more quickly identify experimental conditions that are worthy of further study. These large datasets also open the door to bioinformatics and other data-driven approaches to investigating biological systems. Software, with its ability to quickly digest large volumes of information, would be able to find relationships in the data that would otherwise be difficult for researchers to recognize. Machine learning approaches, in particular, are promising for this purpose. In this way, bioinformatics can provide new questions and new avenues for investigating multicellular systems.

However, the current issue with data being stored in complex, non-standard formats is especially stifling for biological modelling. Computers are totally incapable of processing data like one would find in an academic publication. To leverage data from such a publication would require a person to translate said data into a machine-readable format. This process is time-consuming and increases the likelihood of errors being introduced to the data. To circumvent this obstacle, any universal standard for describing multicellular systems would need to be both human and machine-readable. This would empower modelling projects, and enable data to be searched through, parsed, and analyzed using standard software. Opening up the field of multicellular biology to computational efforts like this would have profound effects, as described above.

An open-source, universal standard that attempts to meet this need is the MultiCellular Data Standard (MultiCellDS). MultiCellDS is described as a "a community-developed standard to functionally describe cell phenotypes with contextual information from the microenvironment". MultiCellDS uses the eXtensible Markup Language (XML), a hierarchical data format that reflects the hierarchical nature of biological systems, to store data in a machine and human-readable way.

data element	a specific measurement (e.g., the radius of a cell nucleus); the smallest unit of data in MultiCellDS
XML	An extensible markup language (similar to HTML used for webpages), which can order data element hierarchically to match their relationships in biology
XML schema	A template for XML data, which describes the allowed data elements and how they can be arranged
metadata	Data about the data: extra information such as units, scale, uncertainty, or provenance
ontology	A controlled dictionary of allowed terms
OWL	A standardized ontology file, in the "web ontology language" format
provenance	The history of who created data, who revised it, who maintains it, and where it is published.
MultiCellDS	multicellular data standard
MultiCellDB	MultiCellDS database
MultiCellXML	MultiCellDS data, written in an XML format
phenotype dataset	A collection of cell phenotype and other measurements, in a single microenvironmental context
digital cell line	A collection of phenotype datasets and key metadata for a single biological cell line or type
digital snapshot	A readout of all cells, their phenotypes, and the microenvironment at a single time
collection	A logical grouping of one or more digital cell lines, digital snapshots, collections, or a combination of these

Figure 2 - MultiCellDS Glossary of Terms (Friedman, Anderson et al. 2016)

MultiCellDS formats data in three ways. *Digital cell lines (DCLs)* are a hierarchical representation of phenotypic data for a given cell type, *digital snapshots* record the spatial information for the cells and their microenvironment, and *collections* are groupings of *DCLs* and *digital snapshots* based on study or other factors.

The ultimate goal of the MultiCellDS project is to allow for the creation of databases, tools, and models that will elevate the current paradigms of multicellular systems biology.

One such model that has emerged is PhysiCell, which the authors describe as an "an open source, agent-based modeling framework for 3-D multicellular simulations" (Ghaffarizadeh, Heiland et al. 2018). PhysiCell allows users to model hundreds of

thousands of cells at once, or perhaps millions, depending on the computational power being used. Accurate multicellular systems are simulated using sub-models that simulate "cell fluid and solid volume changes, cycle progression, apoptosis, necrosis, mechanics, and motility". Since PhysiCell was developed in part by researchers involved with the MultiCellDS project, PhysiCell utilizes a modified, compact version of the MultiCellular Data Standard. The PhysiCell version of the MultiCellular Data Standard stores references to external files, such as MATLAB files, in "custom" data elements in a way that would ideally not be allowed in the standard.

microenvironment domain

```
        <cell_population type="individual">
            <custom>
                <simplified_data type="matlab" source="BioFVM">
                    <filename>./output00000010_cells.mat</filename>
                </simplified_data>
                <simplified_data type="matlab" source="PhysiCell">
                    <labels>
                        <label index="0" size="1">ID</label>
                        <label index="1" size="3">position</label>
                        <label index="4" size="1">total_volume</label>
                        <label index="5" size="1">cell_type</label>
                        <label index="6" size="1">cycle_model</label>
                        <label index="7" size="1">current_phase</label>
                        <label index="8" size="1">elapsed_time_in_phase</label>
                        <label index="9" size="1">nuclear_volume</label>
                        <label index="10" size="1">cytoplasmic_volume</label>
                        <label index="11" size="1">fluid_fraction</label>
                        <label index="12" size="1">calcified_fraction</label>
                        <label index="13" size="3">orientation</label>
                        <label index="16" size="1">polarity</label>
                        <label index="17" size="1">migration_speed</label>
                        <label index="18" size="3">motility_vector</label>
                        <label index="21" size="1">migration_bias</label>
                        <label index="22" size="3">motility_bias_direction</label>
                        <label index="25" size="1">persistence_time</label>
                        <label index="26" size="1">motility_reserved</label>
                        <label index="27" size="1">oncoprotein</label>
                    </labels>
                    <filename>./output00000010_cells_physicell.mat</filename>
                </simplified_data>
            </custom>
        </cell_population>
    </cell_populations>
```

Figure 3 - A PhysiCell File With A "custom" Element, Which Contains Children Not Allowed by The Standard

Currently, any data within a “custom” element is not checked to see if it fits within the standard, so any new elements that PhysiCell files nests within “custom” elements, namely “simplified_data” and its children, must be added to the standard. Additionally, a script for updating old PhysiCell files is required so that they match the updated MultiCellular Data Standard.

II. PROCEDURE

A. Project Overview

At the start of this project, there was already a version of the MultiCellular Data Standard available that was mostly suitable for validating PhysiCell files. The exception to this was the data being stored in the "simplified_data" element, that was found nested in "custom" elements. This was undesirable because, as mentioned above, the standard does not check the children of "custom" elements for content validation. Therefore, the key objective of updating the standard is to incorporate the children of the "custom" data elements, found in PhysiCell files, into the MultiCellular Data Standard.

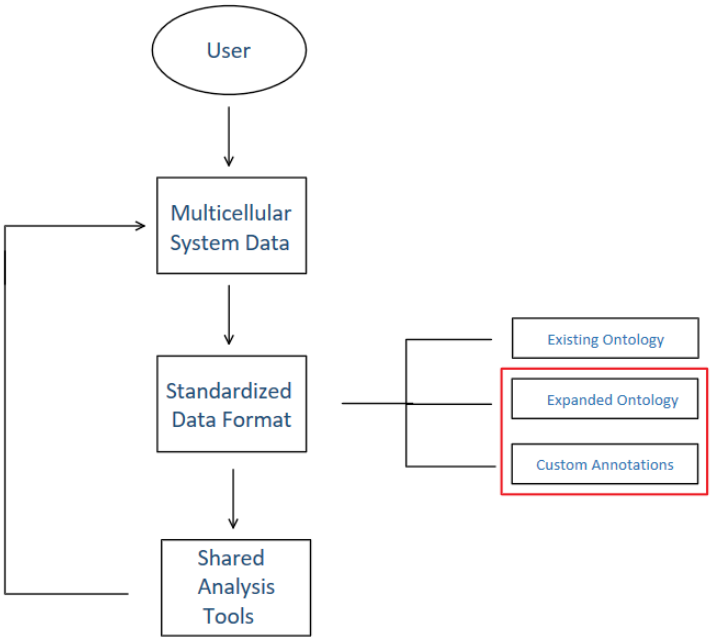

Figure 4 - Flowchart Showing Focus of Project

As shown in figure 4 above, the focus of the project is to expand the existing ontology within the standard to allow for the removal of the "custom" data elements from

PhysiCell files. This will allow the child elements within the “custom” tags to be validated using the standard.

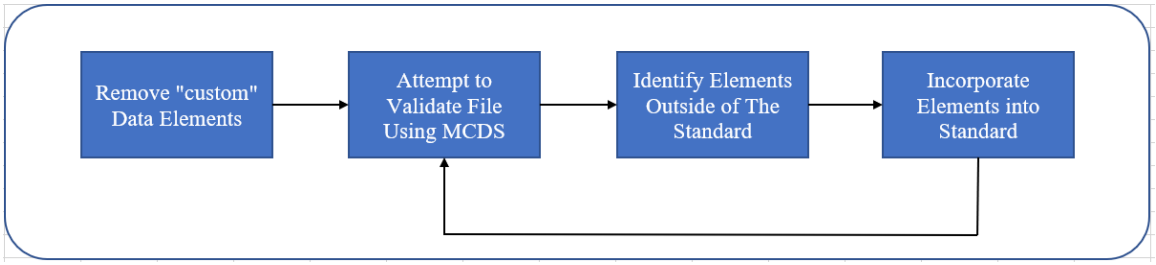

Figure 5 - Flowchart Showing How Standard Was Updated

A program called Oxygen XML Editor was used to open and edit .xsd and .xml files and to validate .xml files against the standard.

A secondary goal of the project is to create a method for updating old PhysiCell files to match the new standard. Since the current PhysiCell paradigm is to store references to external data within “custom” elements, once the standard no longer accepts those elements, PhysiCell files will not be considered valid within the updated standard. To remedy this, a script for editing PhysiCell files is required. Such a script would need to remove the “custom” data elements and its children that violate the standard, while replacing them with the proper tags.

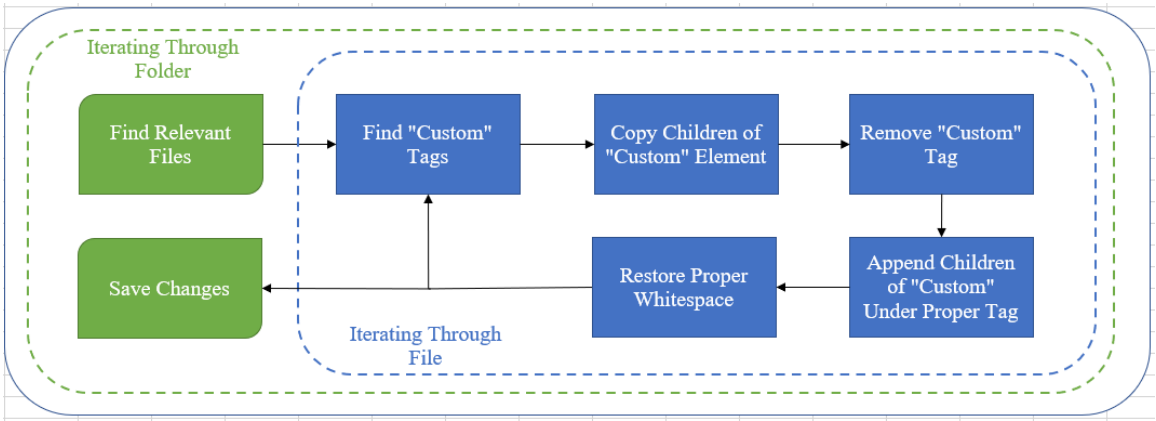

Figure 6 - Flowchart Showing How Update Script Was Written

This script was written in Python 3.9 and requires the xml.etree and lxml libraries to operate, as well as the standard Python modules glob and os. The script was developed in PyCharm.

B. Updating Standard

XML schema definitions (XSDs) are documents that can be written in a hierarchical format, similar to XML data, that are used to define the appropriate structure and format of an XML document. For the purpose of MultiCellDS, these schema documents are used to ensure that data is formatted properly, that necessary metadata is present, and that the data are of the correct type. The schema can be used to "validate" XML documents, which will flag all the errors and sections of the document that do not match the standard. This process is crucial for any effort to create a repository, as data that do not follow the standard would be incompatible with any tools meant to work with MultiCellDS data. XML schemas define the allowed content of an XML file, which are composed of elements. The schema defines what elements can be found in a document, the type of content each element can contain, which attributes each element can have, and parent-child relationships between elements, among other things.

The MultiCellular Data Standard is written as multiple XML Schema Definition (.xsd) files, which reference each other for element type definitions. MultiCellDS is written using the venetian blind model, which utilizes local element declarations, but has globally defined element types (Walmsley 2013). This increases reusability since type definitions can be used in multiple places and are used in the definitions of other complex types. Another advantage is that locally defined elements can have the same name but different attributes, which increases flexibility.

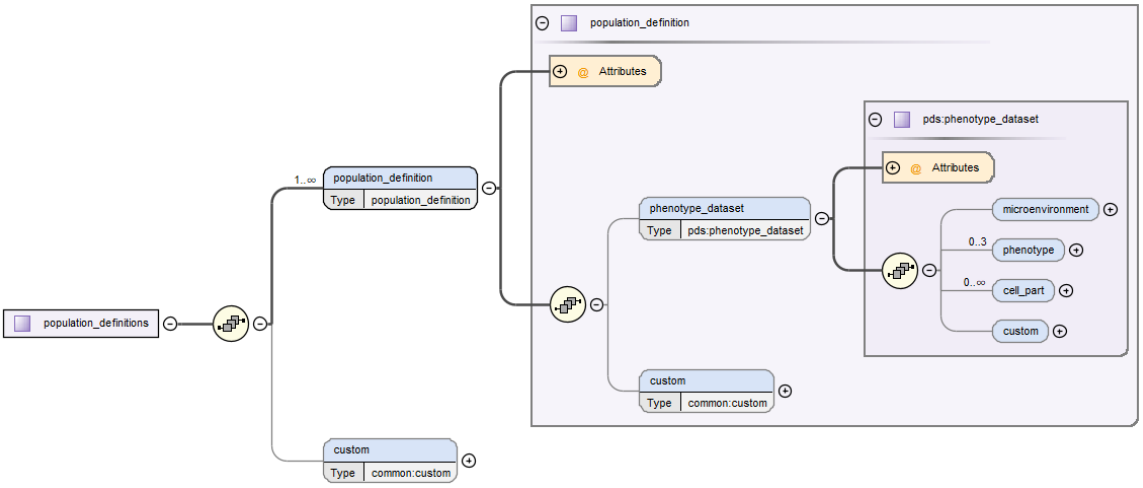

Figure 7 - Example of The Hierarchical nature of XML Schema.

Figure 7 shows both the hierarchical nature of XML schemas and XML data. The elements “microenvironment”, “phenotype”, “cell_part”, and “custom” are all included in the type-definition of “phenotype_dataset”, which is in turn included in the type-definition of “population_definition”. This example also demonstrated the nature of the Venetian Blind model, as each child element of “population_definition” and “phenotype_dataset” is defined separately and not within the type-definition of its parent, meaning they are defined globally.

The validation functionality built into Oxygen XML Editor was used to identify portions of PhysiCell files that did not fit into the existing MultiCellular Data Standard. Sample PhysiCell data were generated using example projects, which are included as a part of a standard PhysiCell download. As can be seen in figure 8, after removing the “custom” data tag, Oxygen flags the “simplified_data” tag because it does not fit within the standard.

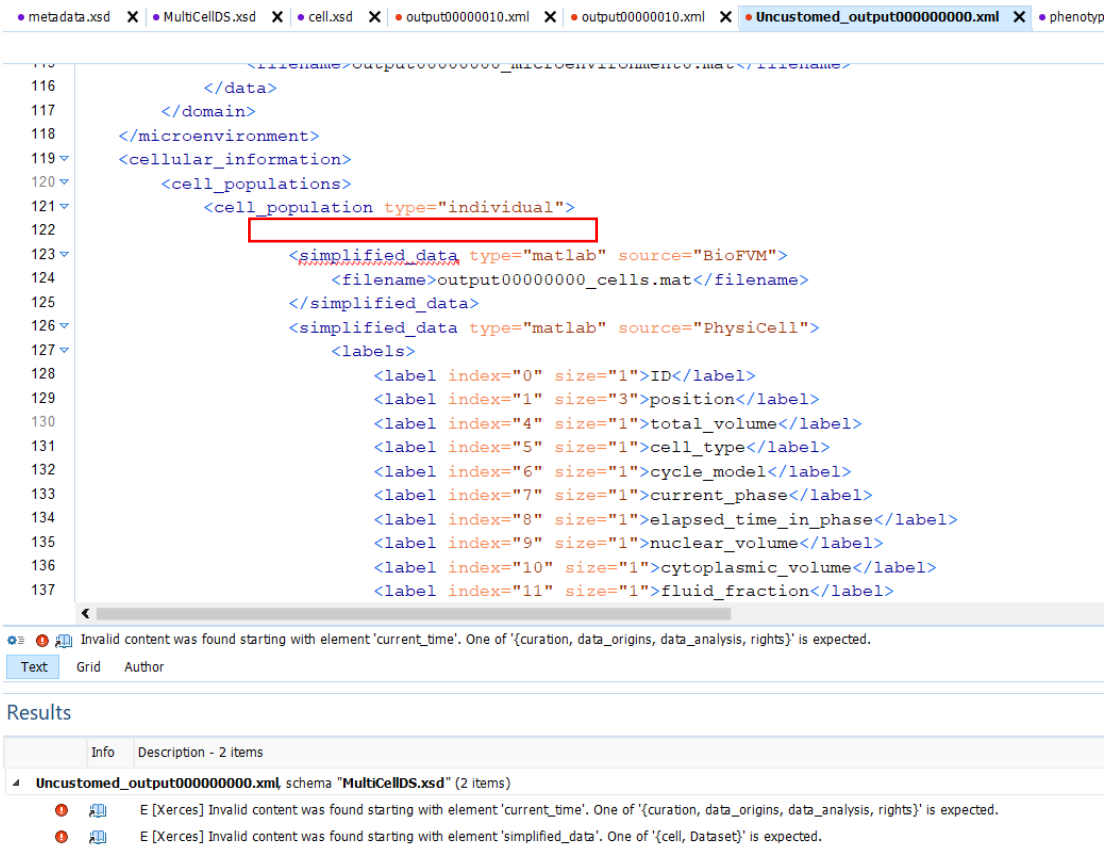

Figure 8 – Example validation of a PhysiCell File. The red box shows where the "custom" data tag used to be.

This functionality was used to identify elements that were not part of the current standard and would need to be added. The "simplified_data" element needed to be incorporated into the standard, meaning that rules governing its content and position were needed. To accomplish this, the definition of the "cell_population" complex type was changed to allow it to contain a child element called "simplified_data" (later renamed to "Dataset") and a global type-definition for "Dataset" was needed for the standard. The child of "Dataset", "labels", and its child "label" were also given global type definitions, in keeping with the Venetian Blind Model. Making these changes allowed for the data in figure 8, after changing the "simplified_data" tag to "Dataset" to be successfully validated using the MultiCellular Data Standard.

However, not all changes were focused on adding new elements to the standard to reflect those used in PhysiCell data. Other changes were focused on providing more complete metadata to describe the new elements. For example, “Dataset” was given an optional “ID” attribute, which can be used to store a unique identifier that links it to specific metadata. This was necessary because multiple instances of “Dataset” can exist in one document, and each instance of “Dataset” may require multiple metadata elements to fully describe its origin.

The MultiCellular Data Standard is written hierarchically in XML Schema Definition (.xsd) files. The master file, MCDS.xsd, references several other external files, which in turn reference each other.

C. Writing Script

Due to the changes made to the standard, it was necessary to create a script for updating PhysiCell files to match the new standard. Since most of the changes made to the standard were to allow for the addition of new elements and metadata, the script only needed to remove the custom data elements and replace them with the proper tag. PyCharm was used to develop the script and, as when editing the standard, Oxygen XML was used to open the XML files being modified. Using these programs allowed for rapid iteration of the script as its effects on PhysiCell files could be observed immediately.

The script, written in Python, takes an input directory and finds all the XML files in said directory. Then, the script searches through each file and finds the “custom” data elements that are no longer allowed by the standard. Once it finds such an element, it copies its child elements and assigns that data to a variable, then deletes the “custom” tag. The children of the “custom” element are the appended back into the proper position

under the “Dataset” tag. Finally, the script restores the proper spacing and saves the edited file.

Figure 9 - PhysiCell File Before Update script

Figure 10 - PhysiCell file after update script.

III. RESULTS AND DISCUSSION OF RESULTS

A. Validation Testing

The PhysiCell update script was validated using data produced by sample PhysiCell models, tabulated below. The "control" version of each file had its "custom" data tags removed so that its contents could be validated against the standard. The "experimental" versions were the updated file after being run through the update script. Validation was considered successful if no errors were found relating to the contents of the "Dataset" element and if the whitespace/indentation of each element was properly aligned.

```
<cell_populations>
    <cell_population type="individual">
            <simplified_data type="matlab" source="BioFVM">
                <filename>output00000000_cells.mat</filename>
            </simplified_data>
            <simplified_data type="matlab" source="PhysiCell">
                <labels>
                    <label index="0" size="1">ID</label>
                    <label index="1" size="3">position</label>
                    <label index="4" size="1">total_volume</label>
                    <label index="5" size="1">cell_type</label>
                    <label index="6" size="1">cycle_model</label>
                    <label index="7" size="1">current_phase</label>
                    <label index="8" size="1">elapsed_time_in_phase</label>
                    <label index="9" size="1">nuclear_volume</label>
                    <label index="10" size="1">cytoplasmic_volume</label>
                    <label index="11" size="1">fluid_fraction</label>
                    <label index="12" size="1">calcified_fraction</label>
                    <label index="13" size="3">orientation</label>
                    <label index="16" size="1">polarity</label>
                    <label index="17" size="1">migration_speed</label>
                    <label index="18" size="3">motility_vector</label>
                    <label index="21" size="1">migration_bias</label>
                    <label index="22" size="3">motility_bias_direction</label>
                    <label index="25" size="1">persistence_time</label>
                    <label index="26" size="1">motility_reserved</label>
                    <label index="27" size="1">receptor</label>
                    <label index="28" size="1">elastic coefficient</label>
                </labels>
                <filename>output00000000_cells_physicell.mat</filename>
            </simplified_data>
    </cell_population>
</cell_populations>
```

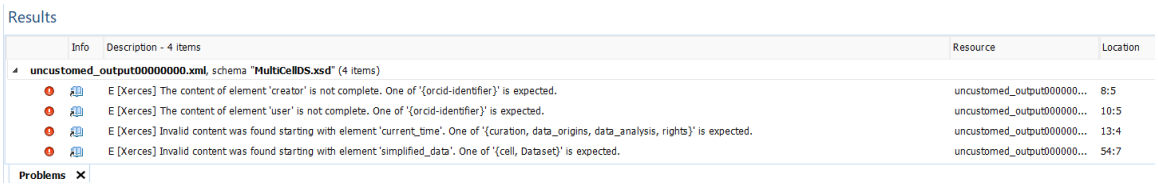

Figure 11 - Validation of an "uncustomed" version

```
<cell_populations>
    <cell_population type="individual">
        <Dataset type="matlab" source="BioFVM">
            <filename>output00000000_cells.mat</filename>
        </Dataset>
        <Dataset type="matlab" source="PhysiCell">
            <labels>
                <label index="0" size="1">ID</label>
                <label index="1" size="3">position</label>
                <label index="4" size="1">total_volume</label>
                <label index="5" size="1">cell_type</label>
                <label index="6" size="1">cycle_model</label>
                <label index="7" size="1">current_phase</label>
                <label index="8" size="1">elapsed_time_in_phase</label>
                <label index="9" size="1">nuclear_volume</label>
                <label index="10" size="1">cytoplasmic_volume</label>
                <label index="11" size="1">fluid_fraction</label>
                <label index="12" size="1">calcified_fraction</label>
                <label index="13" size="3">orientation</label>
                <label index="16" size="1">polarity</label>
                <label index="17" size="1">migration_speed</label>
                <label index="18" size="3">motility_vector</label>
                <label index="21" size="1">migration_bias</label>
                <label index="22" size="3">motility_bias_direction</label>
                <label index="25" size="1">persistence_time</label>
                <label index="26" size="1">motility_reserved</label>
                <label index="27" size="1">receptor</label>
                <label index="28" size="1">elastic coefficient</label>
            </labels>
            <filename>output00000000_cells_physicell.mat</filename>
        </Dataset>
    </cell_population>
</cell_populations>
```

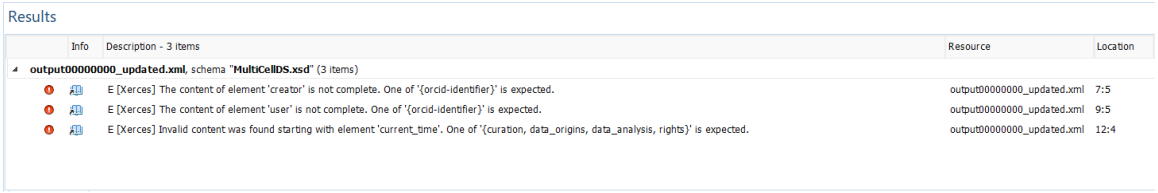

Figure 12 - Validation of Updated Version

Figures 11 and 12 show an example of a successful validation. The results section shows the errors present, or the sections of the document that do not fit the standard. Although both versions have errors relating to missing information, specifically ORCID identifiers,

the updated version does not have the error relating to the "simplified_data" element. Additionally, the updated version retains the proper spacing and whitespace. Data from five different models were used to validate the script, and the results are tabulated below.

Model Name	**Author**	**Successfully Validated?**
Biorobots	Macklin et al.	Yes
Cancer biorobots	Macklin et al.	Yes
Heterogeneity	Macklin et al.	Yes
Cancer immune	Macklin et al.	Yes
pc4covid-19	Macklin et al.	Yes

Table 1 - Validation Testing Results.

As can be seen in the results tabulated above, the update script was able to successfully update data files from each PhysiCell model. This means that the updated versions did not flag errors related to the content of the "custom" tags.

IV. CONCLUSIONS

The goals of updating the standard to match the PhysiCell “compact” format and to create a script for updating deprecated PhysiCell files were successfully met. The compact branch of the MultiCellular Data Standard can now reflect the data structures found in PhysiCell files that reference external data and provide information for interpreting said data. New elements have been added to the standard to store these data and the metadata has been restructured to allow for accurate documentation for the provenance of these data. These updates are an important step in the development of this standard, as they allow the standard to apply to an important use-case. These changes reflect the necessity to expand the standard to enable new uses as those needs arise. By accommodating new use-cases the standard can become more widely adopted. Popularity is crucial for any universal data standard because its utility lies in its ubiquity.

Additionally, the update script developed is capable of converting PhysiCell files into a form that is compliant with the updated standard. This capability is important for facilitating the adoption of the MultiCellular Data Standard for use in PhysiCell models. Modelers will be able to use the script to update their old files, instead of changing their models and re-running them to produce updated versions. This script will ease the transition to the use of MultiCellDS for PhysiCell modelers and complement the updated standard for an important use-case.

V. RECOMMENDATIONS

There are several ways in which the existing MultiCellDS project and the work presented in this paper could be built upon. Firstly, there are ways in which the standard itself could be improved. The way that the standard currently deals with missing metadata, such as missing ORCID identifiers for researchers who produced data, is by flagging it as incorrect, much like it would for data that is formatted incorrectly. The issue of missing metadata is entirely different from extraneous or incorrectly formatted data and should be treated differently. A more helpful approach would be to provide a more descriptive warning about the missing metadata, rather than simply flagging the empty element as an error.

Secondly, the script for updating old PhysiCell files will likely require modification and improvement moving forward. Admittedly, there was not a great variety of data available for testing and validating the script functionality. It is possible that as more data files from different PhysiCell models are run through the script, errors will be encountered due to the presence of previously unencountered elements or formatting.

Finally, the standard must be updated periodically to accommodate the needs of clinicians, researchers, and modelers that utilize the MultiCellular Data Standard. Advancements in research techniques or new use cases will inevitably necessitate the expansion or adjustment of the existing standard to allow for fully descriptive data. This

project is an example of such a situation. As a universal standard for describing multicellular systems biology, it is paramount to remain abreast of the needs of the field to ensure that all use cases are catered to.

REFERENCES CITED

- Friedman, S. H., et al. (2016). "MultiCellDS: a community-developed standard for curating microenvironment-dependent multicellular data." bioRxiv: 090456.
- Ghaffarizadeh, A., et al. (2018). "PhysiCell: An open source physics-based cell simulator for 3-D multicellular systems." PLOS Computational Biology **14**(2): e1005991.
- Macklin, P. (2019). "Key challenges facing data-driven multicellular systems biology." GigaScience **8**(10).
- Walmsley, P. (2013). Definitive XML Schema. Upper Saddle River, N.J., Prentice Hall.

APPENDIX I.

MultiCellDS Compact Branch Commits		
Commit Name	**Date**	**Description**
Commit e0fbb7f1	March 10th, 2021	This commit includes changes intended to allow a particular file produced in PhysiCell, found here: https://gitlab.com/MultiCellDS/MultiCellDS/-/issues/14 to be validated by the MCDS. The sample.zip file contains a data file called output00000010.xml, which can now be validated successfully. The main issues were missing instances of elements, such as orcid_identifier, which were previously required.
Commit a80fda11	June 3rd, 2021	I added a simplified_data element and all of the necessary sub-elements to cell.xsd in order to validate the files generated by the pc4covid19 model. I also added some supporting definitions to common.xsd because I couldn't find existing elements that fit.
Commit 397da1ed	June 9th, 2021	Removed data_sources to allow for any sort of string to be input for a data source. Changed name of element simplified_data to Dataset to match MCDI ontology.
Commit 7fd563a4	April 4th, 2021	Added a key to the ID attribute of data_origin and a keyref to Dataset (placeholder name) to allow for multiple instances of Dataset to be uniquely referenced

		in the metadata.
Commit 8149c2a2	August 24th, 2021	Substituted the "software" element in the complextype "data_origin" for a pared-down version that has less information to reduce redundancy. Changed the number of instances of data_origin to be unbounded to allow for multiple origins to be stored for a single datum.
Commit d79630c1	September 8th, 2021	In cell.xsd: made "filename" element in "Dataset" complextype optional. Added a "Binarized_Data" element to "Dataset". In metadata.xsd: moved the "custom" element to the correct position. Made "data_origin_ID_key" a "unique" type instead of a "key" to make it optional. Made a "software_group" group to allow reusability. "software_group" is used in "data_origin_software" and "software".

Table 2 – MultiCellDS compact branch commits.

APPENDIX II.

Update Script Commits		
Commit Name	**Date**	**Description**
Commit ba0c3c5a	October 5th, 2021	Initial commit.
Commit 343d7580	October 9th, 2021	Cleaned up some comments.

Table 3 - Update script commits.

APPENDIX III.

```
import xml.etree.ElementTree as ET
from lxml import etree
import glob
import os



#Setup for opening a folder and making a list of each .xml file
#mypath = input("Enter a folder location")
mypath = 'C:/Users/Reid/Documents/THESIS STUFF/pc4covid19/Sample_Data'
file_list = []
#for file in os.listdir(mypath):
#    if file.endswith('.xml') and not ("updated" in file):
#        file_list += [file]
file_list = glob.glob(os.path.join(mypath, '*.xml'))
#
for file in file_list:
    file_name = os.path.join(mypath, file)
    #parsing xml doc and determining whether it needs to be updated
    NoWhiteSpaceParser = etree.XMLParser(remove_blank_text=True)
    tree = etree.parse(file_name)
    root = tree.getroot()
    print(root.attrib)
    Dataset = ET.Element("Dataset")


    for cell_populations in root.iter("cell_populations"):
        cell_populations.text = '\n'
        for cell_population in
cell_populations.iter('cell_population'):
            #ET.SubElement(cell_population, "Dataset")
            print(cell_population.attrib)
            for simplified_data in
cell_population.iter('simplified_data'):
                Dataset = simplified_data
                Dataset.tag = "Dataset"
                Dataset_tree = Dataset.iter()
                cell_population.text = '\n'
                for element in Dataset.iter():
                    x = next(Dataset_tree, Dataset)

                    #I'm setting a bunch of values to '\n' here to get
rid of some incorrect whitespace
                    if element.tag == 'Dataset':
                        element.text = '\n'
                        element.tail = '\n'
                    elif element.tag == 'filename':
                        element.tail = '\n'
                    elif element.tag == 'label':
                        element.tail = '\n'
                        print(element.tag)
                    elif element.tag == 'labels':
                        element.text = '\n'
```

```
                        element.tail = '\n'
                    # print(repr(element.text))
                    # print(repr(element.tail))
                     if element.tag == 'filename':
                         print('element is file')
                     print(x.tag)

                     #element.tail = element.tail[0:len(element.tail)-2]
                #Dataset.append(a)
                cell_population.append(Dataset)

                #These indents are for adding in the correct whitespace
                etree.indent(cell_populations, space='\t')
                etree.indent(cell_populations, space='\t', level=1)
                etree.indent(cell_populations, space='\t', level=2)

            for custom in cell_population.findall("custom"):
                cell_population.remove(custom)
    if not os.path.isdir(os.path.join(mypath, 'Updated')):
        os.mkdir(os.path.join(mypath, 'Updated'))
    output_path = os.path.join(mypath, 'Updated')
    tree.write(os.path.join(output_path,
file[0+len(mypath)+1:len(file)-4] + '_updated.xml'), pretty_print=True)
```

VITA

Reid Alan Honeycutt graduated with his B.S. in bioengineering from the University of Louisville in May of 2020 and his M.Eng. in bioengineering in December of 2021. He was a recipient of the Governor's Scholarship and the national Merit Scholarship. His previous works include aiding in the development of a right ventricular assist device under Dr. Steven Koenig. Reid plans to apply his skills as a bioengineer in developing new devices and software to elevate the standards of patient care.