Cyberinfrastructure Shell
Core Specification 1.0

March 2008

Contents

Chapter 1

Introduction

The Cyberinfrastructure Shell (CIShell) is an open source, community-driven platform for the integration and utilization of datasets, algorithms, tools, and computing resources. It is built specifically to enable (1) algorithm developers to write and disseminate their algorithms in their favorite programming language while retaining their intellectual rights after distribution; (2) data holders to easily disseminate their data for use by others; (3) application developers to design applications from custom sets of algorithms and datasets that interoperate seamlessly; and ultimately (4) end-users to use datasets and algorithms effectively.

1.1 Acknowledgements

The Cyberinfrastructure Shell was designed and developed at the Cyberinfrastructure for Network Science Center (CNSC) at Indiana University in Bloomington, Indiana. The specification and API was designed and authored by Bruce W. Herr II, but received input from all members of the CIShell team. Important contributors from the CIShell team include Katy Börner (director of CNSC), Weixia Huang, Russell Duhon, Micah Linnemeier, and Timothy Kelley. Much of the design of CIShell draws on previous work by Shashikant Penumarthy, Bruce W. Herr II, and Katy Börner on the Information Visualization Cyberinfrastructure (IVC). Thanks go out to all those who have used or contributed to IVC, CIShell, and the Network Workbench (the first project to use CIShell). Development of the Cyberinfrastructure Shell was funded by two grants from the National Science Foundation, NSF IIS-0238261 and NSF IIS-0513650.

1.2 CIShell Platform Overview

The CIShell Platform consists of Java interface definitions for algorithms, data, services for algorithm developers, and services for application developers. Much of the platform uses metadata and is fully defined.

1.3 Reader Level

This specification is written for the following audiences:

- Java algorithm developers

- Non-Java algorithm developers
- Framework and system service developers (system developers)
- Application developers building on CIShell

The CIShell Specification assumes that the reader has at least one year of practical experience in writing Java programs. CIShell is built to run on the OSGi Service Platform Release 4[1] and thus a working knowledge of OSGi is expected. OSGi (and thus CIShell) is highly dynamic and must be taken into consideration when developing anything on CIShell.
Non-Java algorithm developers may not need to know any Java and should be mainly concerned with the metadata definitions for algorithms and data. They may also need to be aware of OSGi and the other services CIShell provides, but more than likely will not directly interact with them.

1.4 Conventions and Terms

In this specification, algorithms are referred to in three different contexts. An abstract algorithm is the pure idea of the algorithm with no actual source code. It is a series of steps sometimes put into pseudo-code and often published in academic journals. An `Algorithm` with a capital A refers to the Java class called Algorithm. And finally, an algorithm with a lowercase A refers to the bundle of code and metadata that encompasses an algorithm written to work with the CIShell Platform. This includes the implementation of `AlgorithmFactory` and `Algorithm`, and the metadata, files, and other code that go into an OSGi bundle.
All other conventions and terms are exactly the same as from OSGi's Core Specification, section 1.4.

1.5 Version Information

This is the first release of the CIShell Platform Specification. All packages are at 1.0 for this release. Subsequent releases may increase the version number of specific packages if changes have been made.

Item	**Package**	**Version**
Framework Specification	org.cishell.framework	Version 1.0
Algorithm Specification	org.cishell.framework.algorithm	Version 1.0
Data Specification	org.cishell.framework.data	Version 1.0
User Adjustable Preferences Specification	org.cishell.framework.userprefs	Version 1.0
Data Conversion Service Specification	org.cishell.service.conversion	Version 1.0
GUI Builder Service Specification	org.cishell.service.guibuilder	Version 1.0
Data Manager Application Service Specification	org.cishell.app.datamanager	Version 1.0
Scheduler Application Service Specification	org.cishell.app.scheduler	Version 1.0

Table 1.1: Packages and Versions

[1] http://www.osgi.org/Release4/Download

Chapter 2

Framework API

Version 1.0

2.1 Introduction

The `org.cishell.framework` package and subpackages define the core of CIShell. The key components being algorithms, data, and CIShell service access.

2.1.1 Entities

- *AlgorithmFactory* - The service interface for algorithms. A factory class which creates an `Algorithm` for execution from input data.
- *Algorithm* - The interface for the code execution part of the algorithm.
- *AlgorithmProperty* - The interface which provides string constants for an algorithm's service metadata.
- *ParameterMutator* - The interface an `AlgorithmFactory` extends to provide the ability to add, remove, or modify its input parameters specification (see section 2.5) before being transformed into a form for user input.
- *DataValidator* - The interface an `AlgorithmFactory` extends to provide additional data validation in addition to the data format validation that an application should provide ahead of time.
- *ProgressTrackable* - The interface an `Algorithm` extends to allow for more detailed monitoring and control of an `Algorithm`'s progress while executing.
- *ProgressMonitor* - The interface for a class to be passed in to a `ProgressTrackable Algorithm` so that the `Algorithm` can be controlled and provide information on its progress while executing.
- *Data* - The interface used to pass data (other than input parameters) and its metadata between algorithms.
- *BasicData* - A simple implementation of the `Data` interface.
- *DataProperty* - The interface which provides string constants for `Data` metadata.

- *CIShellContext* - The interface for a class to be passed in to an `AlgorithmFactory` for use in gaining access to standard CIShell services.
- *LocalCIShellContext* - A simple implementation of the `CIShellContext` interface which pulls CIShell services from the OSGi service registry.

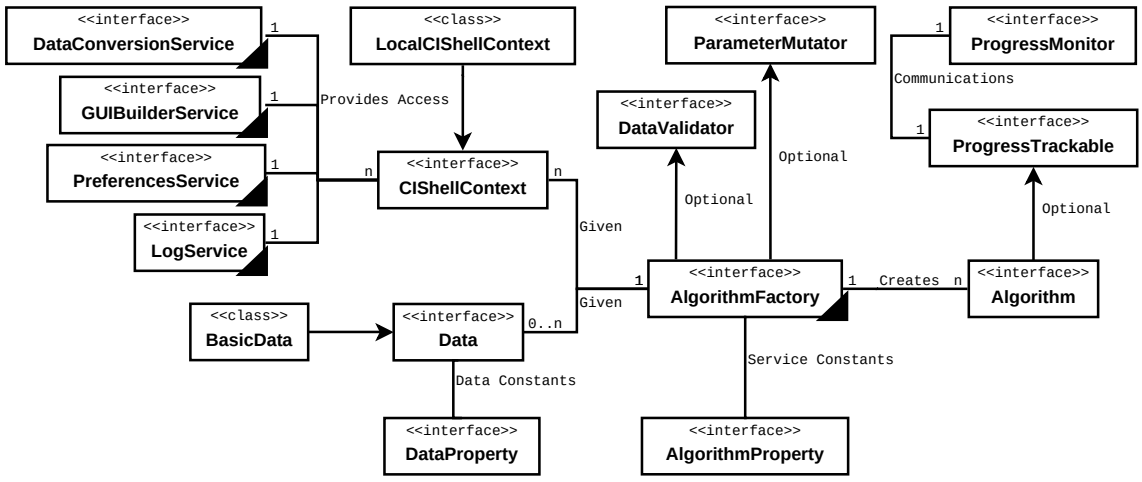

Figure 2.1: org.cishell.framework Class Diagram

2.1.2 Operations

The algorithm developer should implement algorithms as described in this specification. The system developer will provide the services required by CIShell in OSGi's service registry. Application developers will provide everything else, orchestrating the passing of information between algorithms.

2.2 OSGi Dependencies

CIShell is built to be run in a fully compliant OSGi Service Platform R4 implementation. In addition to the base OSGi implementation, several optional OSGi services are required to be available in a fully compliant CIShell implementation. Each additional required service is described in the OSGi Service Platform Service Compendium R4[1].

Required Services

Metatype Service is described in OSGi section 105 "Metatype Service Specification." This service is used by CIShell to find `MetaTypeProviders` for user interface specification, user-adjustable preferences, and input parameters. It also provides an XML format for automatically generating `MetaTypeProviders` which the `MetaTypeService` harvests for use.

Log Service is described in OSGi section 101 "Log Service Specification." This service is used as a universal logging system for algorithms and services. See chapter 5 for more details.

[1] http://www.osgi.org/Release4/Download

Preferences Service is described in OSGi section 106 "Preferences Service Specification." This service is used as a universal preference storage system for algorithms and services. See chapter 6 for more details.

Configuration Admin Service is described in OSGi section 104 "Configuration Admin Service Specification." This service is used as a manager/provider of configuration information for bundles and services. It is useful for meeting the User Adjustable Preferences (section 2.6) requirements.

Recommended Services

Declarative Services is described in OSGi section 112 "Declarative Services Specification." This service can be used by CIShell algorithms to simplify algorithm service registration and for finding necessary auxilary services.

2.3 Algorithm Specification

Version 1.0

2.3.1 Introduction

The CIShell Platform has been specifically designed around the idea of the algorithm. It is the central and most important concept. Algorithms are fully defined and self-contained bits of execution. They can do many things from data conversion, data analysis, and can even spawn whole external programs if needed. Algorithms are well defined black boxes in that their input and output is specified in each algorithm's service metadata and associated `MetaTypeProvider`. Other than that, CIShell makes no attempt to understand the algorithm.

Essentials

- *Application Independence* - Algorithms must be usable in a wide variety of contexts and should not be tied to any one CIShell environment or front end where possible.
- *User Interface Independence* - Algorithms should not have to tie themselves to a single UI where possible.
- *Black Box Algorithms* - Algorithms are black boxes whose possible interactions are described in metadata.
- *Delayed Execution* - There may be a large delay between an algorithm getting parameters for execution and its actual execution.
- *Remote Execution* - Algorithm interfaces should be designed to facilitate remote execution of algorithms where possible.

Operations

To be recognized by CIShell, an `AlgorithmFactory` must be registered with OSGi's service registry. The service registry requires three things when registering: an interface (`AlgorithmFactory`), an implementation, and a `Dictionary` of metadata. The algorithm

developer provides the implementation and metadata. The metadata helps to differentiate and define the algorithm for search and discovery, see section 2.3.3.
An algorithm defines its inputs in two ways. First, the input data is defined in the algorithm's service metadata. Second, the acceptable user-entered input parameters are defined in a `MetaTypeProvider` and published to the `MetaTypeService`.
Figure 2.2 shows the typical flow of information into and out of an algorithm. First the input parameter specification is pulled from the `MetaTypeService`. If parameters are needed, then a UI is created and user inputs are entered. To create an `Algorithm`, the `AlgorithmFactory` is passed the user-entered parameters, zero or more pieces of data, and a `CIShellContext`. The `Algorithm` is then executed and produces zero or more pieces of data.

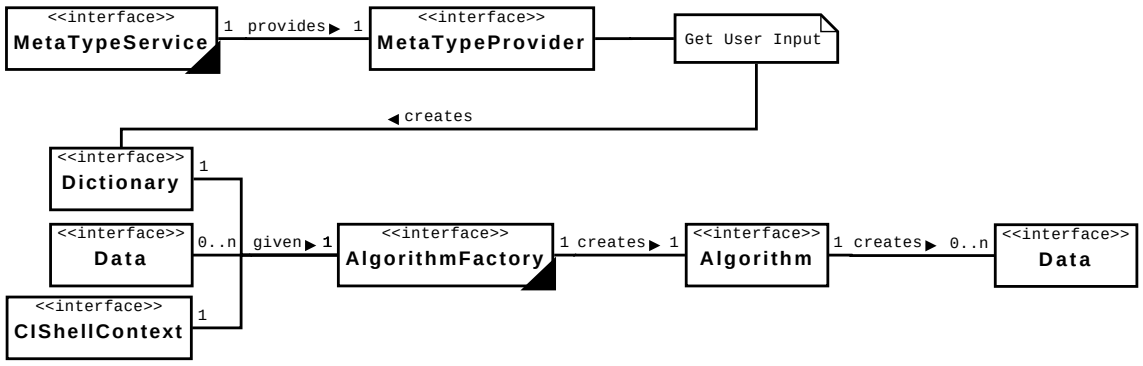

Figure 2.2: Algorithm Execution Workflow

2.3.2 Optional Interfaces

Algorithm developers may augment algorithms with additional interfaces to enhance parts of the execution workflow. See each interface's API documentation for more details.

ParameterMutator An `AlgorithmFactory` can implement the `ParameterMutator` interface to add, remove, or modify an algorithm's input parameters between the time when its `MetaTypeProvider` is pulled from the `MetaTypeService` and when the corresponding UI is shown to the user. This is typically done to customize the input parameters based on the data to be given to the algorithm. See section 2.5 for information on constructing and publishing `MetaTypeProviders`.

DataValidator An `AlgorithmFactory` can implement the `DataValidator` interface to validate the data beyond the data format validation that an application should provide ahead of time.

ProgressTrackable An `Algorithm` can implement `ProgressTrackable` to allow for more detailed monitoring and control of an `Algorithm`'s progress while executing.

2.3.3 Service Metadata

When an algorithm is registered with OSGi's service registry, a `Dictionary` of metadata is provided. Since the algorithm itself is a black box, the metadata is used to provide information about the algorithm. Information such as the format of each `Data` item to be input and output is provided. In addition to the mechanics of the algorithms, data such as the authors, label, urls,

citation references, and description are provided. This metadata can be searched by anyone to find relevant algorithms using OSGi's service registry.
Each standard metadata element required by the CIShell specification is defined below and in the interface `AlgorithmProperty`. It defines each key string and the valid value strings to set in the metadata `Dictionary` when registering an algorithm as a service.

service.pid

A string that uniquely identifies the algorithm. The service.pid should not change between sessions and only one algorithm with a given service.pid should be available in the service registry at any given time. It is recommended to use the Java naming scheme including path for this purpose, i.e., "org.cishell.my.algorithm.MyAlgorithm". This metadata element is defined as part of the OSGi Service Platform Core Specification, section 6.1.12.

Example 1: service.pid = org.my.algorithm.MyAlgorithm

parameters_pid

When this key is not set, an algorithm's user-entered input parameters are assumed to be registered in the `MetaTypeService` with a persistent id equal to the algorithm service's "service.pid". An algorithm can override this by setting this key to their custom persistent id. See section 2.5 for information on creating and publishing input parameters.

Example 1: parameters_pid = org.my.custom.pid.that.i.want.to.use

in_data

Specifies the formats and number of `Data` inputs the algorithm accepts. The string is a comma separated list of data formats as defined in section 2.4. If no `Data` inputs are necessary then the string "null" or not specifying the in_data attribute at all is valid. If any data is optional, prefix the associated format with a "?". When the `AlgorithmFactory.createAlgorithm` method is called with a `Data` array, an optional `Data` element will be `null` if it is not provided.
By using the "+", "*", or "?" prefixes, ambiguities could arise from specifying multiple input formats that could easily fit into two or more of the formats. An algorithm developer should take care when reading in `Data` arrays from possibly ambiguous in_data strings.

Example 1: in_data = null
Example 2: in_data = java.lang.String
Example 3: in_data = java.lang.String, file:text/plain, file:text/xml
Example 4: in_data = file-ext:xml
Example 5: in_data = +file:text/plain, *file:text/xml, ?java.lang.String
Example 6: in_data = ?my.package.SpecialClass

out_data

Specifies the formats and number of `Data` inputs the algorithm produces when successfully executed. The string is a comma separated list of data formats as defined in section 2.4. If no `Data` is output then the string "null" or not specifying the out_data attribute at all is valid. If any data is optional, prefix the associated format with a "?". When producing the `Data` array, an optional `Data` element must be `null` if it is not produced.

Example 1: out_data = null

Example 2: out_data = java.lang.String

Example 3: out_data = java.lang.String, file:text/plain, file:text/xml

Example 4: out_data = file-ext:csv

Example 5: out_data = +file:text/plain, *file:text/xml, ?java.lang.String

Example 6: out_data = ?my.package.SpecialClass

parentage

If this metadata element is used, it defines how the output `Data` produced by the algorithm should be arranged. `Data` items can be given a parent as part of their metadata (which usually means the `Data` was derived from the referenced `Data`). If parentage is set to "default" then each of the algorithm's output `Data` items will have their parent `Data` item set as the first input `Data` item (if applicable) by the CIShell-conforming application. If parentage is set to something else or is not set at all, then it is up to the algorithm to set up these relationships.

Example 1: parentage = default

type

Specifies the type of the algorithm. If no type is set, then it is assumed to be of "Standard Algorithm" type. Which metadata keys to use and their exact meaning varies depending on the type of algorithm. The different algorithm types and their constraints are defined in section 2.3.4.

Example 1: type = converter

Example 2: type = validator

Example 3: type = dataset

remotable

Specifies if the algorithm can be run remotely. An algorithm can be run remotely if it only uses the services provided by the `CIShellContext` and does not create its own non-`GUIBuilderService`-built GUI. Valid strings are "true" or "false". If this metadata element is not set, then it is assumed that it cannot be run remotely.

Example 1: remoteable = true

label

Specifies a human-readable short name for the algorithm. What label is acceptable varies depending on the type of the algorithm.

Example 1: label = Load...

description

Provides more details on the workings of the algorithm. What description is acceptable varies depending on the type of the algorithm.

Example 1: description = Loads selected data into the data manager.

menu_path

Specifies where on the menu an algorithm is to be placed if a menu bar is used. Otherwise, it can act as a primitive hierarchical classification of the algorithm. The string is a "/" separated list with each element in the list getting more specific. The last element in the list specifies a group for grouping algorithms in its final menu. Possible groups include: "additions" for default placement, "start" for being placed at the start of the menu, or "end" for being placed at the end of the menu.

Example 1: menu_path = File/additions

Example 2: menu_path = Analysis/Undirected Networks/start

Example 3: menu_path = Visualization/Networks/end

conversion

For converter algorithms, this metadata element specifies if any data is lost in the conversion. Possible values are "lossy" and "lossless". A description of what type of information is lost should be given in the "description" metadata field.

Example 1: conversion = lossy

authors

A comma separated list of the authors of the abstract algorithm.

Example 1: authors = Bruce W. Herr II

Example 2: authors = Bruce W. Herr II, Weixia Huang, Katy Borner

implementers

A comma separated list of the developers who implemented the algorithm in code.

Example 1: implementers = Bruce W. Herr II

Example 2: implementers = Bruce W. Herr II, Weixia Huang, Katy Borner

integrators

A comma separated list of the developers who integrated the algorithm code as a compliant cishell algorithm.

Example 1: integrators = Bruce W. Herr II

Example 2: integrators = Bruce W. Herr II, Weixia Huang, Katy Borner

documentation_url

A URL to relevant documentation for the algorithm.

Example 1: documentation_url = http://cishell.org/dev/

reference

A formal reference to a paper explaining the abstract algorithm.

> **Example 1:** reference = Herr, Bruce W. II, Huang, Weixia, Penumarthy, Shashikant, Börner, Katy. (2007) Designing Highly Flexible and Usable Cyberinfrastructures for Convergence. In William S. Bainbridge and Mihail C. Roco (Eds.) Progress in Convergence - Technologies for Human Wellbeing. Annals of the New York Academy of Sciences, Boston, MA, volume 1093, pp. 161-179.

reference_url

A URL to a paper explaining the abstract algorithm.

> **Example 1:** reference_url = http://cishell.org/papers/07-cishell.pdf

written_in

A comma separated list of the programming languages used to implement and integrate the algorithm code.

> **Example 1:** written_in = Java
>
> **Example 2:** written_in = Java, C++

2.3.4 Algorithm Types

Introduction

CIShell algorithms are a generic concept which have many uses. In the CIShell Platform, they are used in 4 contexts: as general data-centric algorithms to be used by an end-user (Standard Algorithms), as data converters (Converter Algorithms), as data validators (Validator Algorithms), and as providers of data (Dataset Algorithms). In order to better separate these uses, a lightweight type system has been introduced. The only way to tell the difference between them is by the constraints defined for each algorithm type as defined below.

Base Algorithm Constraints

All conformant algorithms regardless of type, must adhere to the following constraints:

Required:

- The algorithm must be a conformant `AlgorithmFactory` implementation and properly registered as a service.
- The algorithm's service metadata must contain a valid "service.pid".

Optional:

- The algorithm's service metadata should have "remoteable=true" if it meets the requirements of a remoteable algorithm.
- The algorithm's service metadata should have a "label" which is a short human-readable name for the algorithm.
- The algorithm's service metadata should have a "description" explaining what the algorithm does in more detail.
- As much of the informational metadata as possible should be provided. This includes "authors", "implementors", "integrators", "documentation_url", "reference", "reference_url", and "written_in".

Standard Algorithms

Standard CIShell algorithms are the algorithms that most end-users will encounter. A standard algorithm has the following constraints:

Required:

- The algorithm must be a conformant `AlgorithmFactory` implementation and properly registered as a service.
- The algorithm's service metadata must contain a valid "service.pid".
- The algorithm's service metadata must have a "label" which is a short human-readable name for the algorithm. This is typically used to label an algorithm for an end-user to see.
- The algorithm's service metadata must have a "description" explaining what the algorithm does in more detail.
- The algorithm's service metadata must have a "menu_path" which is simultaneously a classification and a location on a GUI's menubar to place the algorithm in. See section 2.3.3 for how to format a "menu_path".

Optional:

- If additional user-entered input parameters are needed, the algorithm should provide a `MetaTypeProvider` published to the `MetaTypeService`.
- The algorithm's service metadata should have "remoteable=true" if it meets the requirements of a remoteable algorithm.
- The algorithm's service metadata should have "parentage=default" if it wishes to use the default `Data` parenting scheme described in section 2.3.3.
- The algorithm's service metadata does not need to have a "type" set.
- As much of the informational metadata as possible should be provided. This includes "authors", "implementors", "integrators", "documentation_url", "reference", "reference_url", and "written_in".

Dataset Algorithms

A dataset algorithm is a custom type of CIShell algorithm for providing pre-generated data for use in the CIShell Platform. Dataset algorithms act just like standard algorithms and have a superset of requirements. CIShell Applications may not even treat them any differently than standard algorithms. A dataset algorithm has the following constraints:

Required:

- The algorithm must be a conformant `AlgorithmFactory` implementation and properly registered as a service.
- The algorithm's service metadata must contain a valid "service.pid".
- The algorithm's service metadata must have a "label" which is a short human-readable name for the dataset being provided. This is typically used to label a dataset for an end-user to see.
- The algorithm's service metadata must have a "description" explaining what the dataset is in more detail.
- The algorithm's service metadata must have a "menu_path" which is simultaneously a classification and a location on a GUI's menubar to place the dataset in. Datasets will typically be in "File/Datasets/additions" See section 2.3.3 for how to format a "menu_path".
- The algorithm's service metadata must have "type=dataset".
- The algorithm's service metadata must have "in_data=null" or not defined at all.
- The algorithm's service metadata must have at least one data item set as its "out_data".

Optional:

- The algorithm's service metadata should have "remoteable=true" if it meets the requirements of a remoteable algorithm.
- As much of the informational metadata as possible should be provided. This includes "authors", "implementors", "integrators", "documentation_url", "reference", and "reference_url".

Converter Algorithms

A converter algorithm is a custom type of CIShell algorithm for converting data of one type to another. Converters are typically leveraged by the `DataConversionService` and are not used directly by end-users. A converter algorithm has the following constraints:

Required:

- The algorithm must be a conformant `AlgorithmFactory` implementation and properly registered as a service.

- The algorithm must take in a single `Data` item and convert the item, producing a single `Data` item. This must be reflected in the algorithm's service metadata where "in_data" and "out_data" have only one data item each.
- The algorithm's service metadata must contain a valid "service.pid".
- The algorithm's service metadata must have "type=converter".
- The algorithm's service metadata must have "conversion=lossy" if data is lost during conversion or "conversion=lossless" if not.
- The algorithm must not require any input parameters. The `Dictionary` passed to the createAlgorithm method will always be empty.

Optional:

- The algorithm's service metadata should have "remoteable=true" if it meets the requirements of a remoteable algorithm.
- The algorithm's service metadata should have a "label" which is a short human-readable name for the converter, usually with the common name of the input data format and output data format.
- The algorithm's service metadata should have a "description" explaining the conversion in more detail, especially what data may be lossed if "conversion=lossy".
- The algorithm's service metadata should have "implementers" filled in accordingly.

Validator Algorithms

A validator algorithm is a custom type of CIShell algorithm which checks either an incoming or outgoing file to be sure it is of the type specified. This is necessary due to the fact that one cannot simply assume based on the file extension what type of file format the data is in. Checking the contents of the file is necessary, especially in the case of multiple file formats for the same file extension (e.g., XML). This type of algorithm is important for reliably bringing in outside data and saving out data from CIShell. A validator algorithm has the following constraints:

Required:

- The algorithm must be a conformant `AlgorithmFactory` implementation and properly registered as a service.
- The algorithm's service metadata must contain an "in_data" and "out_data" with only one data item each with one containing a "file:*mime/type*" format and the other a "file-ext:*file-extension*" depending on the direction of validation. See section 2.4 for data format details.
- The algorithm must take in a single `Data` item and validate the item producing a single `Data` item (with the same data, but changed format) if the file is of the right type. If not, then the algorithm should log (using the CIShellContext-provided `LogService`) what the problem was and must return null. If a problem occurs that is unrelated to the file's format, then an `AlgorithmExecutionException` should be thrown.

- The algorithm must not alter the data. Its only purpose is to validate the proposed incoming or outgoing file.
- The algorithm's service metadata must contain a valid "service.pid".
- The algorithm's service metadata must have "type=validator".
- The algorithm's service metadata must have a "label" which is the common name of the data format being validated.
- The algorithm must not require any input parameters. The `Dictionary` passed to the createAlgorithm method will always be empty.

Optional:

- The algorithm's service metadata should have "remoteable=true" if it meets the requirements of a remoteable algorithm.
- The algorithm's service metadata should have "implementers" filled in accordingly.

2.4 Data Specification

Version 1.0

2.4.1 Introduction

Data to be operated on is passed around in `Data` objects which hold the real data, the data's format, and the data's properties (metadata). The real data can be any Java `Object`. The format is a string describing the 'format' of the data, see next section. Finally, the properties help describe the data. The label to give the data, the parent `Data` object from which it was derived from, and a coarse data type can all be defined in the `Data`'s properties. See the `DataProperty` interface definition for specific properties to use.

2.4.2 Data Format Specification

Data formats are used by algorithms specifying what `Data` items are consumed/producted and `Data` objects must know what the format of its contained real data is. This format is simply a string that says what the format of the real data is. If the real data is a `java.io.File`, use a MIME type[2] prefixed by "file:", i.e., "file:*mime/type*". If the real data is a `java.io.File` known only by file extension (only applicable for validator algorithms), use the format "file-ext:*file-extension*". Otherwise, if the real data is a Java `Object`, use the full Java class as a string, i.e., "java.lang.String" or "my.package.SpecialClass".
To specify that a single `Data` object contains one or more sub-`Data` objects of a single format, prefix the type with a "+". For zero or more, prefix the type with a "*". This corresponds to a `Data` object that is wrapping multiple (zero or one or more depending on the prefix) other `Data` objects of the associated format in a Java array (`Data[]`) stored as its contained real data. This is useful for algorithms that can work on a variable number of `Data` items.

[2]If no official mime type is available for a file format, a made up one can be used, but must still conform to how mime types are constructed, see RFCs 3023 (http://tools.ietf.org/html/rfc3023) and 4288 (http://tools.ietf.org/html/rfc4288).

2.5 User Interface Specification

Version 1.0

2.5.1 Introduction

For many algorithms, just looking at the data given isn't enough. Additional input parameters are often needed to know how to operate on a given piece of data. An algorithm can define what parameters are needed by providing a `MetaTypeProvider`. It defines the types, value range, and textual description of the parameters needed. From this information, a user interface (UI) can be created that asks a user for the data. The `MetaTypeProvider` is not tied to any specific UI, so it can be reused depending on the context (desktop application, web application, command line, etc.).
`MetaTypeProvider` is a Java interface defined in the OSGi R4 Specification Service Compendium as part of the "Metatype Service Specification," section 105. A `MetaTypeProvider` can be thought of as a collection of UIs. Each UI is called an `ObjectClassDefinition`, which provides a UI name and description and is a collection of parameters. Each parameter is an `AttributeDefinition` which includes the type, label, description, default value, and range of valid values. Drop-down boxes can also be defined by using option labels and values with the `AttributeDefinition`. OSGi's documentation should be consulted for more information.

2.5.2 MetaTypeProvider Extensions

Some minor extensions to `MetaTypeProvider` were made to support some use cases. The `MetaTypeProvider` supports several primitive types such as strings, integers, booleans, etc, but several useful types are missing. To support more types, an `AttributeDefinition` (AD) of type "string" has its default value set to a certain string so that the UI builder recognizes this and selects an appropriate widget. When the algorithm receives the user-entered parameters, the associated value will be of type `java.lang.String`, but should contain the correct value as defined below.

file:

An AD with type "string" and default value "file:" will receive a string pointing to the absolute path to the file selected by the end-user.

directory:

An AD with type "string" and default value "directory:" will receive a string pointing to the absolute path to the directory selected by the end-user.

password:

An AD with type "string" and default value "password:" will receive a string corresponding to the entered password.

rgb:

An AD with type "string" and default value "rgb:" will receive a string which is a comma separated list that corresponds to the RGB color values the user chose. Each item in the comma

separated list would be between 0 and 255. The first item would be the red value, second green value, and third blue value.

2.5.3 Publishing MetaTypeProviders

For user-adjustable preferences and algorithm input parameters, a `MetaTypeProvider` is required to be published to the `MetaTypeService`. This can be done in two ways, through code or by a METADATA.XML file.
To publish through code, a few steps must be followed. First, the service to be registered with the OSGi service registry must fully implement `org.osgi.service.cm.ManagedService` and `org.osgi.service.metatype.MetaTypeProvider`. Second, when registering the service, both `ManagedService` and `MetaTypeProvider` must be in the list of interfaces the service implements. If these two things are done, the `MetaTypeService` will notice it and add it to its registry of `MetaTypeProviders`.
The recommended way to publish `MetaTypeProviders` is to publish through a METADATA.XML file. A METADATA.XML file must be included in the algorithm's OSGi bundle in a specific directory, "OSGI-INF/metatype/". The `MetaTypeService` will notice this in the bundle and add it to its registry of `MetaTypeProviders`. See the "Metatype Service Specification" in the OSGi specification for details on the XML format.

2.6 User Adjustable Preferences Specification

Version 1.0

2.6.1 Introduction

The user-adjustable preferences specification defines how any service can publish user-adjustable preferences both globally and locally. In addition to global and local preferences, algorithms can allow the system to allow end-users to adjust the default values for algorithms' user-entered input parameters specification published to the `MetaTypeService`. For storing data that is not directly end-user adjustable, see chapter 6.

2.6.2 Publishing User Adjustable Preferences

Create an ObjectClassDefinition (OCD)

To define parameters that can be adjusted by an end-user, an algorithm developer must first create an `ObjectClassDefinition` which details the parameters to be published. This OCD must be visible to the `MetaTypeService` either through the use of a METADATA.XML file or by the service implementing `MetaTypeProvider` and `ManagedService`. See section 2.5 for more information.

Designate an OCD a Persistent ID (PID)

Then they must designate the `ObjectClassDefinition` a unique persistent id (PID). The PID can be designated in two ways. The simplest way is by following the convention of creating a string with the associated service's "service.pid" and appending either ".prefs.local" or ".prefs.global". The other way is to designate whatever PID the developer wishes and to provide a service property "local_pref_pid" or "global_pref_pid" which is set to whatever PID they chose.

Declare What Preferences are to be Published

To let the system know that you wish to publish preferences, the system properties must contain a "prefs_published" key with zero or more of the following values (separated by commas): "local" for publishing local prefs, "global" for global prefs, and "param-defaults" for algorithm parameter defaults.

Algorithm Parameter Defaults

By publishing algorithm parameter defaults, algorithm developers allow end-users to adjust the default values they see when running their algorithm. This is typically accomplished by wrapping the `MetaTypeProvider` published by the algorithm to the `MetaTypeService` with overridden `AttributeDefinitions` that change their default value. Many systems will have this on by default, but if the "prefs_published" key is set in the algorithm's service metadata and "param-defaults" is not set, then this feature will be disabled for the algorithm.

Receiving Preference Data

To be notified of changes to local or global preferences, the service must implement `org.osgi.service.cm.ManagedService` and set in their service metadata "receive_prefs=true". When either the local or global preferences are updated, the updated method will be passed a `Dictionary` of all of the id/value pairs, including the updated ones. Local preferences will have the same ids as the `AttributeDefinition`s (AD) defined in the associated OCD. The local preferences will also have an additional id "Bundle-Version", which contains the version of the service's associated bundle that was used when the preference data was last updated. Global preferences will have the same ids (plus a "Bundle-Version" id analagous to local preference's) from their OCD's ADs prefixed by the PID of the published global preference. In this way, all global preferences published in the system will be available to anyone receiving preference data. Note that global preferences can be received without publishing preferences.

2.7 org.cishell.framework

Interface **CIShellContext**

The context by which algorithms in the framework can gain access to standard CIShell services. An instantiated CIShellContext must provide access to at least the default services (as of the 1.0 specification, the OSGi LogService, the OSGi PreferencesService, the CIShell defined DataConversionService, and the CIShell defined GUIBuilderService). Other services may be made available through this class, but anything beyond the standard services is not guaranteed.

Declaration

```
public interface CIShellContext
```

Fields

- public static final String DEFAULT_SERVICES
  - Contains an array of the valid strings corresponding to the default services

Methods

- *getService*
  `public java.lang.Object` **getService**`( java.lang.String` **service** `)`
  - **Usage**
    - Locates and returns a standard service given the service name. The service name is generally the full class name of the service interface. For example, `LogService`'s string is `org.osgi.service.log.LogService`.
  - **Parameters**
    - `service` - A string (usually the associated interface's full class name) that specifies the service to retrieve
  - **Returns** - An instantiated version of the service requested

Class **LocalCIShellContext**

A simple implementation of CIShellContext that pulls the CIShell services from the provided BundleContext that all OSGi bundles receive on activation. This was included in the standard API since it will be used frequently by CIShell application developers.
This implementation only returns standard services or the service strings given to it in its constructor.

Declaration

public class LocalCIShellContext
extends java.lang.Object
implements CIShellContext

Constructors

- *LocalCIShellContext*
  `public` **LocalCIShellContext**`( org.osgi.framework.BundleContext` **bContext** `)`
  - **Usage**
    - Initializes the CIShell context based on the provided `BundleContext`
  - **Parameters**
    - `bContext` - The `BundleContext` to use to find the registered standard services
- *LocalCIShellContext*
  `public` **LocalCIShellContext**`( org.osgi.framework.BundleContext` **bContext**`,`
  `java.lang.String[]` **standardServices** `)`

- **Usage**
  - Initializes the CIShell context with a custom set of standard services. Only the service in the array will be allowed to be retrieved from this `CIShellContext`.
- **Parameters**
  - `bContext` - The `BundleContext` to use to find registered standard services
  - `standardServices` - An array of strings specifying the services that are allowed to be retrieved from this class

Methods

- *getService*
  `public java.lang.Object` **`getService`**`( java.lang.String` **`service`** `)`
  - **See Also**
    - `org.cishell.framework.CIShellContext.getService( java.lang.String )` (in 2.7, page 20)

2.8 org.cishell.framework.algorithm

Interface **Algorithm**

A class which executes some arbitrary code and optionally returns any data produced. What happens when the execute method is run is entirely up to the Algorithm developer. Algorithms should be primed with whatever data is needed, usually by its associated AlgorithmFactory, before execution. This allows an Algorithm to be set up, then scheduled for later execution.

Declaration

public interface Algorithm

Methods

- *execute*
  `public Data` **`execute`**`( )`
  - **Usage**
    - Executes and optionally returns a Data array
  - **Returns** - A Data array that was created. `null` is ONLY acceptable when the algorithms out_data is null.
  - **Exceptions**
    - `org.cishell.framework.algorithm.AlgorithmExecutionException` - An exception has occured while executing the algorithm. This exception should have a user-comprehendable message if at all possible.

Interface **AlgorithmFactory**

A service interface for creating Algorithms to be executed. An algorithm developer must create an implementation of this interface and register it (along with some standard metadata about the algorithm, defined in the AlgorithmProperty class) in the OSGi service registry. If the algorithm requires input in addition to the raw data provided, a MetaTypeProvider must be published to OSGi's MetaTypeService (usually through a METADATA.XML file in the algorithm's bundle).
See the **CIShell Specification 1.0** at http://cishell.org/dev/docs/spec/cishell-spec-1.0.pdf for documentation on the full requirements for algorithm creation.

Declaration

```
public interface AlgorithmFactory
```

Methods

- *createAlgorithm*
  `public Algorithm` **createAlgorithm**`( Data[]` **data**`, java.util.Dictionary` **parameters**`, CIShellContext` **context** `)`
  - **Usage**
    - Creates an Algorithm to be executed
  - **Parameters**
    - `data` - The data to be given to the Algorithm to process. Some Algorithms may ignore this value. The order and type of data given are specified in the service dictionary (the 'in_data' key) when registered as a service in OSGi.
    - `parameters` - A set of key-value pairs that were created based on the associated input specification published to the MetaTypeService
    - `context` - The context by which the Algorithm can gain access to standard CIShell services
  - **Returns** - An `Algorithm` primed for execution

Interface **AlgorithmProperty**

A standard set of properties and values used for creating a service metadata Dictionary that is provided when registering an AlgorithmFactory with the OSGi service registry.
See the **CIShell Specification 1.0** at http://cishell.org/dev/docs/spec/cishell-spec-1.0.pdf for documentation on each property.

Declaration

```
public interface AlgorithmProperty
```

- public static final String IN_DATA = “in_data”

  –

- public static final String OUT_DATA = “out_data”

  –

- public static final String NULL_DATA = “null”

  –

- public static final String PARAMETERS_PID = “parameters_pid”

  –

- public static final String PARENTAGE = “parentage”

  –

- public static final String DEFAULT_PARENTAGE = “default”

  –

- public static final String ALGORITHM_TYPE = “type”

  –

- public static final String TYPE_CONVERTER = “converter”

  –

- public static final String TYPE_VALIDATOR = “validator”

  –

- public static final String TYPE_DATASET = “dataset”

  –

- public static final String REMOTEABLE = “remoteable”

  –

- public static final String REMOTE = “remote”

  –

- public static final String LABEL = “label”

  –

- public static final String DESCRIPTION = “description”

  –

- public static final String MENU_PATH = “menu_path”

  –

- public static final String ADDITIONS_GROUP = “additions”
  –
- public static final String START_GROUP = “start”
  –
- public static final String END_GROUP = “end”
  –
- public static final String CONVERSION = “conversion”
  –
- public static final String LOSSY = “lossy”
  –
- public static final String LOSSLESS = “lossless”
  –
- public static final String AUTHORS = “authors”
  –
- public static final String IMPLEMENTERS = “implementers”
  –
- public static final String INTEGRATORS = “integrators”
  –
- public static final String DOCUMENTATION_URL = “documentation_url”
  –
- public static final String REFERENCE = “reference”
  –
- public static final String REFERENCE_URL = “reference_url”
  –
- public static final String WRITTEN_IN = “written_in”
  –

Interface **DataValidator**

An additional interface an AlgorithmFactory can implement that allows for further data validation beyond what is provided in the service dictionary’s in_data/out_data specifications. This is useful in cases where an algorithm expects a certain type of data, but must make additional checks. For example, if an algorithm only worked on symmetric matrices, this interface would check the data ahead of time to ensure that the given matrix was in fact a symmetric matrix.

Declaration

public interface DataValidator

Methods

- *validate*
  `public java.lang.String` **validate**`( Data[]` **data** `)`
  - **Usage**
    - Validates the given data that is proposed to be given to the algorithm. It can return three different values:

`null`	No validation present
`""`	The data is valid
`"..."`	A localized description of why its invalid

  - **Parameters**
    - `data` - The proposed data that may be given to create an Algorithm
  - **Returns** - `null`, "", or another string

Interface **ParameterMutator**

An additional interface an AlgorithmFactory can implement that allows for adding, modifying, or removing input parameters before being shown to the end-user for input. This interface is often implemented by algorithms that wish to customize the user interface based on the actual input data.

Declaration

public interface ParameterMutator

Methods

- *mutateParameters*
  `public org.osgi.service.metatype.ObjectClassDefinition` **mutateParameters**`(`
  `Data[]` **data**`, org.osgi.service.metatype.ObjectClassDefinition` **parameters** `)`
  - **Usage**
    - Adds, modifies, or removes Algorithm parameters (AttributeDefinitions) from a given ObjectClassDefinition returning either the same (if no changes are made) input or a new, mutated version of the input
  - **Parameters**
    - `data` - An optional argument, the Data array that will be given to this class to create an Algorithm with the createAlgorithm method. Applications that don't know the Data array that is going to be used ahead of time can give a `null` value.

* `parameters` - A set of AttributeDefinitions which define the algorithm's input parameters
- **Returns** - An OSGi ObjectClassDefinition that defines the parameters needed by the Algorithm this class creates

Interface **ProgressMonitor**

A class to monitor the progress of an algorithm. It allows for notification of progress, notification of cancellation, notification of pausing, and description of current work during execution. Except for the setter methods, the methods are generally only called by the algorithm with the CIShell application providing the progress monitor implementation.

Declaration

```
public interface ProgressMonitor
```

Fields

- public static final ProgressMonitor NULL_MONITOR
  - A monitor with empty methods for use by algorithms when no ProgressMonitor has been given to it. This helps to eliminate spurious `null` checks to ensure the progress monitor is not `null`.
- public static final int WORK_TRACKABLE = 2
  - Capability constant specifying that this algorithm can update its work progress (value is 1<<1)
- public static final int CANCELLABLE = 4
  - Capability constant specifying that this algorithm can be cancelled (value is 1<<2)
- public static final int PAUSEABLE = 8
  - Capability constant specifying that this algorithm can be paused (value is 1<<3)

Methods

- *describeWork*
  `public void` **describeWork**`( java.lang.String` **currentWork** `)`
  - **Usage**
    * Method to describe what the algorithm is currently doing for the benefit of the users of the algorithm as it progresses during execution
  - **Parameters**
    * `currentWork` - A short description of the current work the algorithm is doing

---

- *done*
  `public void` **done**`( )`
  - **Usage**
    - The algorithm is finished executing

- *isCanceled*
  `public boolean` **isCanceled**`( )`
  - **Usage**
    - Returns whether cancellation of algorithm execution is requested. An algorithm that can be cancelled should poll this method when convenient to see if it should cancel.
  - **Returns** - Whether cancellation of algorithm execution is requested

---

- *isPaused*
  `public boolean` **isPaused**`( )`
  - **Usage**
    - Returns whether pausing of algorithm execution is requested. An algorithm that can be paused should poll this method when convenient to see if it should pause.
  - **Returns** - Whether pausing of algorithm execution is requested

---

- *setCanceled*
  `public void` **setCanceled**`( boolean` **value** `)`
  - **Usage**
    - Sets or clears a flag for cancellation of this algorithm's execution. An algorithm developer can ignore or clear this flag if it cannot stop midstream. This is one of the methods that can be called by something other than the algorithm.
  - **Parameters**
    - `value` - Set or clear the cancellation request

- *setPaused*
  `public void` **setPaused**`( boolean` **value** `)`
  - **Usage**
    - Sets or clears a flag for pausing of this algorithm's execution. An algorithm developer can ignore or clear this flag if it cannot pause midstream. This is one of the methods that can be called by something other than the algorithm.
  - **Parameters**
    - `value` - Set or clear the pause request

- *start*
  `public void` **start**`( int` **capabilities**`, int` **totalWorkUnits** `)`
  - **Usage**

    - Notifies the start of execution of the algorithm in addition to revealing how many work units will be used
  - **Parameters**
    - `capabilities` - An OR'ed int that tells the monitor what the algorithm is capable of with respect to the monitor. The OR'ed values are taken from the int constants specified in this interface.
    - `totalWorkUnits` - The number of work units, -1 if the algorithm does not provide progress information

---

- *worked*
  `public void` **worked**`( int` **work** `)`
  - **Usage**
    - Notifies that a certain number of units of work has been completed
  - **Parameters**
    - `work` - The number of units of work completed since last notification

Interface **ProgressTrackable**

An additional interface an Algorithm can implement that allows for monitoring of progress, process cancellation, and current work description. This was not included in the `Algorithm` interface because many of the algorithms will not be able to support these features (especially the algorithms that are wrapping executable programs). Even algorithms that do implement this interface do not have to provide all of the features. For instance, an algorithm may only support progress notification and not cancellation.

Declaration

```
public interface ProgressTrackable
```

Methods

- *getProgressMonitor*
  `public ProgressMonitor` **getProgressMonitor**`( )`
  - **Usage**
    - Returns the progress monitor currently in use, or `null` if no monitor has been set
  - **Returns** - The current progress monitor, or `null` if there isn't one set

---

- *setProgressMonitor*
  `public void` **setProgressMonitor**`( ProgressMonitor` **monitor** `)`
  - **Usage**
    - Sets the progress monitor this algorithm is to use. This method should be called before an algorithm is executed. If this method is not set prior to execution, the algorithm must run without it.

– **Parameters**
  * `monitor` - The monitor the algorithm is to use

Class **AlgorithmExecutionException**

An exception which is thrown when an error occurs in the process of executing an Algorithm

Declaration

public class AlgorithmExecutionException
extends java.lang.Exception

Constructors

- *AlgorithmExecutionException*
  `public` **AlgorithmExecutionException**`( java.lang.String` **message** `)`
- *AlgorithmExecutionException*
  `public` **AlgorithmExecutionException**`( java.lang.String` **message**`,`
  `java.lang.Throwable` **exception** `)`
- *AlgorithmExecutionException*
  `public` **AlgorithmExecutionException**`( java.lang.Throwable` **exception** `)`

Methods inherited from class `java.lang.Exception`

Methods inherited from class `java.lang.Throwable`

- *fillInStackTrace*
  `public synchronized native java.lang.Throwable` **fillInStackTrace**`( )`
- *getCause*
  `public java.lang.Throwable` **getCause**`( )`
- *getLocalizedMessage*
  `public java.lang.String` **getLocalizedMessage**`( )`
- *getMessage*
  `public java.lang.String` **getMessage**`( )`
- *getStackTrace*
  `public java.lang.StackTraceElement[]` **getStackTrace**`( )`
- *initCause*
  `public synchronized java.lang.Throwable` **initCause**`( java.lang.Throwable` **arg0** `)`
- *printStackTrace*
  `public void` **printStackTrace**`( )`

- *printStackTrace*
  `public void` **printStackTrace**`( java.io.PrintStream` **arg0** `)`
- *printStackTrace*
  `public void` **printStackTrace**`( java.io.PrintWriter` **arg0** `)`
- *setStackTrace*
  `public void` **setStackTrace**`( java.lang.StackTraceElement[]` **arg0** `)`
- *toString*
  `public java.lang.String` **toString**`( )`

2.9 org.cishell.framework.data

Interface Data

A class that contains data, its format, and its metadata. This class is used to pass data between algorithms and is what algorithms optionally create when executed.

Declaration

public interface Data

Methods

- *getData*
  `public java.lang.Object` **getData**`( )`
  - **Usage**
    - Returns the data stored in this Data object
  - **Returns** - The data (a Java object)
- *getFormat*
  `public java.lang.String` **getFormat**`( )`
  - **Usage**
    - Returns the format of the encapsulated data. If the data is a File, then this method returns what MIME type it is with "file:" prepended (eg. file:text/plain). Otherwise, the string returned should be the Java class it represents. For algorithms this format should be the same as their OUT_DATA property.
  - **Returns** - The main format of the data
- *getMetadata*
  `public java.util.Dictionary` **getMetadata**`( )`
  - **Usage**
    - Returns the metadata associated with the data stored in this Data object. Standard keys and values are in the DataProperty interface.
  - **Returns** - The data's metadata

Interface **DataProperty**

Standard property keys and values to use when creating metadata for a Data object

Declaration

```
public interface DataProperty
```

Fields

- public static final String LABEL = "Label"
  - The label to give the Data object if displayed. The type associated with this property is of type String.
- public static final String SHORT_LABEL = "Short_Label"
  - A short label to give the Data object for shorter displays. It is recommended to keep the string length below 20 characters. This will often be used for recommended file names when saving the data to disk. The type associated with this property is of type String.
- public static final String PARENT = "Parent"
  - The parent Data object of the Data object. This is used when a Data object is derived from another Data object to show the hierarchical relationship between them. This property can be null, signifying that the Data object was not derived from any other Data object, such as when loading a new Data object from a file. The type associated with this property is of type Data.
- public static final String TYPE = "Type"
  - The general type of the Data object. Various standard types are created as constants with name *_TYPE from this class. These can be used, or new types can be introduced as needed. The type associated with this property is of type String.
- public static final String MODIFIED = "Modified"
  - Flag to determine if the Data object has been modified and not saved since the modification. This is used to do things like notify the user before they exit that a modified Data object exists and ask if they want to save it. The type associated with this property is of type Boolean.
- public static final String MATRIX_TYPE = "Matrix"
  - Says this data model is abstractly a matrix
- public static final String NETWORK_TYPE = "Network"
  - Says this data model is abstractly a network

- public static final String TABLE_TYPE = "Table"
  - Says this data model is abstractly a table
- public static final String TREE_TYPE = "Tree"
  - Says this data model is abstractly a tree
- public static final String OTHER_TYPE = "Unknown"
  - Says this data model is abstractly an unknown type
- public static final String TEXT_TYPE = "Text"
  - Says this data model is abstractly a plain text file
- public static final String PLOT_TYPE = "Plot"
  - Says this data model is abstractly a data plot

Class **BasicData**

A basic implementation of Data. This class was included since a simple implementation of Data will be used quite often in both application and algorithm code.

Declaration

public class BasicData
extends java.lang.Object
implements Data

Constructors

- *BasicData*
  `public` **BasicData**`( java.util.Dictionary` **properties**`, java.lang.Object` **data**`, java.lang.String` **format** `)`
  - **Usage**
    - Creates a Data object with the given data and metadata Dictionary
  - **Parameters**
    - `properties` - The metadata about the data
    - `data` - The data being wrapped

- *BasicData*
  `public` **BasicData**`( java.lang.Object` **data**`, java.lang.String` **format** `)`
  - **Usage**
    - Creates a Data object with the given data and an empty metadata Dictionary
  - **Parameters**
    - `data` - The data being wrapped

METHODS

- *getData*
  `public java.lang.Object` **getData**`( )`
  - **See Also**
    - `org.cishell.framework.data.Data.getData( )`

- *getFormat*
  `public java.lang.String` **getFormat**`( )`
  - **See Also**
    - `org.cishell.framework.data.Data.getFormat( )`

- *getMetadata*
  `public java.util.Dictionary` **getMetadata**`( )`
  - **See Also**
    - `org.cishell.framework.data.Data.getMetadata( )`

2.10 org.cishell.framework.userprefs

INTERFACE UserPrefsProperty

A standard set of properties and values to be placed in a service's metadata Dictionary when registering a service with the OSGi service registry for the purpose of publishing and receiving user-adjustable preferences.
See the **CIShell Specification 1.0** at http://cishell.org/dev/docs/spec/cishell-spec-1.0.pdf for information on publishing user-adjustable preferences.

DECLARATION

```
public interface UserPrefsProperty
```

FIELDS

- public static final String LOCAL_PREFS_OCD_SUFFIX = ".prefs.local"
  - The suffix to add to the service's PID for generating a local preferences PID when using the standard naming convention
- public static final String GLOBAL_PREFS_OCD_SUFFIX = ".prefs.global"
  - The suffix to add to the service's PID for generating a global preferences PID when using the standard naming convention

- public static final String PARAM_PREFS_OCD_SUFFIX = “”
  - The suffix to add to the service's PID for an Algorithm's user-entered input parameters PID when using the standard naming convention
- public static final String LOCAL_PREFS_PID = “local_prefs_pid”
  - The key for specifying a local preferences PID. Only use this when not following the standard naming convention.
- public static final String GLOBAL_PREFS_PID = “global_prefs_pid”
  - The key for specifying a global preferences PID. Only use this when not following the standard naming convention.
- public static final String PREFS_PUBLISHED_KEY = “prefs_published”
  - The key for specifying what types of preferences are published
- public static final String PUBLISH_LOCAL_PREFS_VALUE = “local”
  - The value for specifying that local preferences are to be published
- public static final String PUBLISH_GLOBAL_PREFS_VALUE = “global”
  - The value for specifying that global preferences are to be published
- public static final String PUBLISH_PARAM_DEFAULT_PREFS_VALUE = “param-defaults”
  - The value for specifying that an Algorithm's user-entered input parameter defaults may be adjusted by the user
- public static final String RECEIVE_PREFS_KEY = “receive_prefs”
  - The key for declaring a need to receive preferences. ”true” and ”false” are the possible associated values.
- public static final String LOCAL_PREFS_CONF_SUFFIX = “”
  - The suffix to add to the service's PID for getting the local preferences directly from the ConfigurationAdmin (not recommended)
- public static final String GLOBAL_PREFS_CONF_SUFFIX = “.prefs.global”
  - The suffix to add to the service's PID for getting the global preferences directly from the ConfigurationAdmin (not recommended)
- public static final String PARAM_PREFS_CONF_SUFFIX = “.prefs.params”
  - The suffix to add to the service's PID for getting an Algorithm's user-entered input parameter defaults that have been user-adjusted directly from the ConfigurationAdmin (not recommended)
- public static final String BUNDLE_VERSION_KEY = “Bundle-Version”
  - A key set in each configuration object which states the Bundle-Version of the service when it was last updated

Chapter 3

Data Conversion Service Specification

Version 1.0

3.1 Introduction

A conscious design decision was made for CIShell not to enforce a central data model/format that all algorithms have to work with. Instead, an algorithm expresses the data format of each data item coming into and out of the algorithm in its service metadata. It is the job of the code calling the algorithm to get the data in the right format before calling the algorithm. The Data Conversion Service is used here to simplify the process of converting data.

3.1.1 Entities

- *DataConversionService* - The service interface for converting data to different formats.
- *Converter Algorithm* - A special type of algorithm, defined on page 14, which converts data from one format to another.
- *Converter* - The interface for a wrapped set of converter algorithms returned by the `DataConversionService` that will convert data from one format to another.

3.2 Data Conversion Service

The Data Conversion Service provides unified access to converter algorithms. `DataConversionService` system developers may choose not to leverage converter algorithms, but this is ill-advised. Also, good implementations will take advantage of the nature of converter algorithms to allow for more than just single hop conversions. Since all converter algorithms specify a single data object in and a single data object out, a graph can be constructed where nodes are the data formats and edges are the converters. Using this directed graph, when a conversion between data formats is requested, the `DataConversionService` will choose the shortest path of converters to do the conversion. A hypothetical conversion graph is illustrated in figure 3.1.

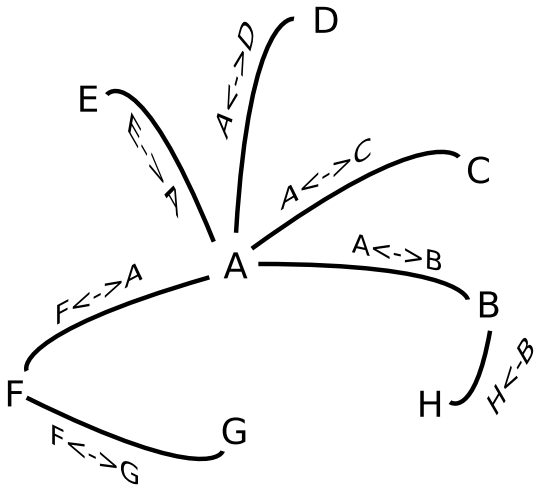

Figure 3.1: A Conversion Graph

3.3 org.cishell.service.conversion

Interface **Converter**

A class for converting Data objects

Declaration

public interface Converter

Methods

- *convert*
  `public Data` **convert**`( Data` **data** `)`
  - **Usage**
    - Uses this Converter to convert the given Data object to a new format. This is a convenience method that uses this Converter to convert a Data object of the corrent format to a Data object of the defined output format.
  - **Parameters**
    - `data` - The Data object with compatible format
  - **Returns** - A Data object of correct output format
  - **Exceptions**
    - `org.cishell.service.conversion.ConversionException` - If the data conversion fails while converting

- *getAlgorithmFactory*
  `public AlgorithmFactory` **getAlgorithmFactory**`( )`

- **Usage**
  - Returns the AlgorithmFactory that can be invoked to convert a given Data object of the correct input format (as specified in the Dictionary from getProperties()) to a Data object of the correct output format
- **Returns** - The AlgorithmFactory to do the converting

- *getConverterChain*
  `public org.osgi.framework.ServiceReference[] getConverterChain( )`
  - **Usage**
    - Returns an array of ServiceReferences to converter algorithms in the order in which they will be called when converting a Data object
  - **Returns** - An array of ServiceReferences to converter algorithms to be used

---

- *getProperties*
  `public java.util.Dictionary getProperties( )`
  - **Usage**
    - Get properties of the Converter (same as algorithm service properties). It is a set of properties that correspond to the AlgorithmPropertys properties. The IN_DATA and OUT_DATA properties are guaranteed to be set in this Dictionary.
  - **Returns** - A set of properties describing the converter (including its in and out data)

Interface **DataConversionService**

A service for converting data to different formats. This service should utilize the pool of AlgorithmFactory services which have registered with the OSGi service registry and specified in its service dictionary that they are a converter. A converter will specify what data format it takes in ('in_data'), what it converts it to ('out_data'), and whether any information will be lost in the conversion ('conversion'='lossless'—'lossy'). Using this and other standard algorithm properties, a DataConversionService will try and find the fastest, most efficient way to convert from one format to another.

Declaration

```
public interface DataConversionService
```

Methods

- *convert*
  `public Data convert( Data data, java.lang.String outFormat )`
  - **Usage**
    - Tries to convert a given Data object to the specified output format
  - **Parameters**
    - `data` - The Data to convert

    - `outFormat` - The format of the Data object to be returned
  - **Returns** - A Data object with the specified output format
  - **Exceptions**
    - `org.cishell.service.conversion.ConversionException` - If the data conversion fails while converting

- *findConverters*
  `public Converter` **findConverters**`( Data` **data**`, java.lang.String` **outFormat** `)`
  - **Usage**
    - Tries to find all the converters that can be used to transform the given Data object to the specified output format
  - **Parameters**
    - `data` - The Data object to convert
    - `outFormat` - The output format to convert to
  - **Returns** - An array of Converters that can convert the given Data object to the specified output format

---

- *findConverters*
  `public Converter` **findConverters**`( java.lang.String` **inFormat**`, java.lang.String` **outFormat** `)`
  - **Usage**
    - Finds converters from one format to another if at all possible. The returned Converters, which may be a composite of multiple algorithms, will take a Data object of the specified `inFormat` and convert it to a Data object of type `outFormat`.
  - **Parameters**
    - `inFormat` - The type of Data object to be converted. This String should be formatted in the same way as an algorithm's AlgorithmProperty.IN_DATA.
    - `outFormat` - The type of Data object that should be produced. This String should be formatted in the same way as an algorithm's AlgorithmProperty.OUT_DATA.
  - **Returns** - An array of Converters that can convert a Data object of the given inFormat to the specified outFormat

Class **ConversionException**

An exception which is thrown when an error occurs in the process of data conversion

Declaration

public class ConversionException
extends java.lang.Exception

CONSTRUCTORS

- *ConversionException*
  `public` **ConversionException**`( java.lang.String` **message** `)`
- *ConversionException*
  `public` **ConversionException**`( java.lang.String` **message**`, java.lang.Throwable` **exception** `)`
- *ConversionException*
  `public` **ConversionException**`( java.lang.Throwable` **exception** `)`

METHODS INHERITED FROM CLASS `java.lang.Exception`

METHODS INHERITED FROM CLASS `java.lang.Throwable`

- *fillInStackTrace*
  `public synchronized native java.lang.Throwable` **fillInStackTrace**`( )`
- *getCause*
  `public java.lang.Throwable` **getCause**`( )`
- *getLocalizedMessage*
  `public java.lang.String` **getLocalizedMessage**`( )`
- *getMessage*
  `public java.lang.String` **getMessage**`( )`
- *getStackTrace*
  `public java.lang.StackTraceElement[]` **getStackTrace**`( )`
- *initCause*
  `public synchronized java.lang.Throwable` **initCause**`( java.lang.Throwable` **arg0** `)`
- *printStackTrace*
  `public void` **printStackTrace**`( )`
- *printStackTrace*
  `public void` **printStackTrace**`( java.io.PrintStream` **arg0** `)`
- *printStackTrace*
  `public void` **printStackTrace**`( java.io.PrintWriter` **arg0** `)`
- *setStackTrace*
  `public void` **setStackTrace**`( java.lang.StackTraceElement[]` **arg0** `)`
- *toString*
  `public java.lang.String` **toString**`( )`

Chapter 4

GUI Builder Service Specification

Version 1.0

4.1 Introduction

The GUI Builder Service provides a user-interface-agnostic solution to create UIs for simple user input. The UIs are built from the user interface specification provided by `MetaTypeProvider` and requires no UI coding to be done other than providing an implementation of `MetaTypeProvider`. Information on creating these classes can be found in section 2.5. In addition, simple methods for creating warnings, pop-ups, and simple yes/no dialog boxes are provided by the GUI Builder Service. The GUI creation workflow is illustrated in figure 4.1.

4.1.1 Entities

- *GUIBuilderService* - The Service interface for creating user interfaces and dialog boxes.
- *MetaTypeProvider* - The interface for creating simple user interface specifications.
- *GUI* - The interface for controlling the `GUIBuilderService` generated user interface.
- *SelectionListener* - The interface to listen for events generated by the user's interaction with the UI.

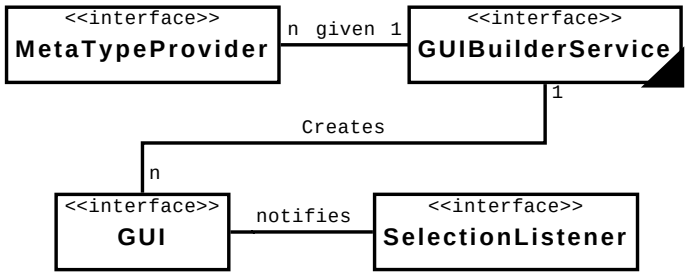

Figure 4.1: GUI Creation Workflow

4.2 org.cishell.service.guibuilder

Interface GUI

A simple GUI for user interaction. A single SelectionListener can be set to be informed when the user enters information and hits Ok.

Declaration

public interface GUI

Methods

- *close*
  `public void` **close**`( )`
  - **Usage**
    - Closes the GUI

- *isClosed*
  `public boolean` **isClosed**`( )`
  - **Usage**
    - Returns if the GUI is closed
  - **Returns** - If the GUI has been closed or not

- *open*
  `public void` **open**`( )`
  - **Usage**
    - Opens the GUI and shows it to the user

- *openAndWait*
  `public java.util.Dictionary` **openAndWait**`( )`
  - **Usage**
    - Pops up this GUI, gets data from the user, and returns what they entered. This is a convenience method that first opens the GUI, then shows the GUI to the user, who then enters in the needed information, which is then taken and put into a Dictionary, and is given to this method's caller.
  - **Returns** - The data the user entered or `null` if the operation was cancelled

---

- *setSelectionListener*
  `public void` **setSelectionListener**`( SelectionListener` **listener** `)`
  - **Usage**

    - Sets the selection listener to be informed when the user finishes entering information and hits 'Ok' or cancels
  - **Parameters**
    - `listener` - The listener to notify

Interface **GUIBuilderService**

A service for creating simple GUIs for user interaction. This service provides several methods for popping up dialog boxes to get or give very simple information and a more flexible way to create GUIs using a standard OSGi MetaTypeProvider. The MetaTypeProvider basically lists what input is needed (String, Integer, Float, etc...), a description of the input, and a way to validate input.
See the **CIShell Specification 1.0** at http://cishell.org/dev/docs/spec/cishell-spec-1.0.pdf for documentation on creating GUIs with this service.
Algorithm writers are encouraged to use this service if they need to get additional input from the user rather than creating their own GUI. This is to ensure a consistent user input method and so that the GUI can easily be routed to the user when running remotely.

Declaration

```
public interface GUIBuilderService
```

Methods

- *createGUI*
  `public GUI` **createGUI**`( java.lang.String` **id**`, org.osgi.service.metatype.MetaTypeProvider` **parameters** `)`
  - **Usage**
    - Creates a GUI for user interaction
  - **Parameters**
    - `id` - The id to use to get the correct ObjectClassDefinition from the provided MetaTypeProvider
    - `parameters` - Provides the parameters needed to get information from the user
  - **Returns** - The created GUI

- *createGUIandWait*
  `public java.util.Dictionary` **createGUIandWait**`( java.lang.String` **id**`, org.osgi.service.metatype.MetaTypeProvider` **parameters** `)`
  - **Usage**
    - Creates a GUI, gets data from the user, and returns what they entered. This is a convenience method that first creates a GUI from the provided MetaTypeProvider, then pops the GUI up to the user, who then enters in the needed information, which is then taken and put into a Dictionary, and is given to this method's caller.
  - **Parameters**

    * `id` - The id to use to get the correct ObjectClassDefinition from the provided MetaTypeProvider
    * `parameters` - Provides the parameters needed to get information from the user
  - **Returns** - The data the user entered or `null` if the operation was cancelled

---

- *showConfirm*
  `public boolean` **showConfirm**`( java.lang.String` **title**`, java.lang.String` **message**`, java.lang.String` **detail** `)`
  - **Usage**
    * Pops up a confirmation box to the user with an 'Ok' and 'Cancel' button
  - **Parameters**
    * `title` - The title of the pop-up
    * `message` - The message to display
    * `detail` - Additional details
  - **Returns** - If they clicked "Ok," `true`, otherwise `false`

- *showError*
  `public void` **showError**`( java.lang.String` **title**`, java.lang.String` **message**`, java.lang.String` **detail** `)`
  - **Usage**
    * Pops up an error box to the user. This should only be used sparingly. Algorithms should try to use the LogService instead.
  - **Parameters**
    * `title` - The title of the pop-up
    * `message` - The message to display
    * `detail` - Additional details

- *showError*
  `public void` **showError**`( java.lang.String` **title**`, java.lang.String` **message**`, java.lang.Throwable` **error** `)`
  - **Usage**
    * Pops up an error box to the user. This should only be used sparingly. Algorithms should try to use the LogService instead.
  - **Parameters**
    * `title` - The title of the pop-up
    * `message` - The message to display
    * `error` - The actual exception that was thrown

- *showInformation*
  `public void` **showInformation**`( java.lang.String` **title**`, java.lang.String` **message**`, java.lang.String` **detail** `)`
  - **Usage**
    * Pops up an information box to the user. This should only be used sparingly. Algorithms should try to use the LogService instead.
  - **Parameters**

    * `title` - The title of the pop-up
    * `message` - The message to display
    * `detail` - Additional details

- *showQuestion*
  `public boolean` **showQuestion**`( java.lang.String` **title**`, java.lang.String` **message**`, java.lang.String` **detail** `)`
  - **Usage**
    * Pops up a question box to the user with a 'Yes' and 'No' button
  - **Parameters**
    * `title` - The title of the pop-up
    * `message` - The question to display
    * `detail` - Additional details
  - **Returns** - If they clicked "Yes," `true`, otherwise `false`

- *showWarning*
  `public void` **showWarning**`( java.lang.String` **title**`, java.lang.String` **message**`, java.lang.String` **detail** `)`
  - **Usage**
    * Pops up a warning box to the user. This should only be used sparingly. Algorithms should try to use the LogService instead.
  - **Parameters**
    * `title` - The title of the pop-up
    * `message` - The message to display
    * `detail` - Additional details

Interface **SelectionListener**

A listener that is notified when all values entered by a GUI user have been validated and they have clicked 'Ok' to proceed or if the operation was cancelled

Declaration

```
public interface SelectionListener
```

Methods

- *cancelled*
  `public void` **cancelled**`( )`
  - **Usage**
    * Notification that the user cancelled the operation

- *hitOk*
  `public void` **hitOk**`( java.util.Dictionary` **valuesEntered** `)`
  - **Usage**
    - Notification that the user hit 'Ok' along with the data they entered
  - **Parameters**
    - `valuesEntered` - The data the user entered

Chapter 5

Log Service Specification

Version 1.3

5.1 Introduction

CIShell requires OSGi's standard `LogService` to be installed in a CIShell-compliant system. This gives both the algorithms and the applications built on CIShell a standard logging sytem to use. This service has not been extended or modified. More information about the `LogService` is available in the OSGi Service Platform Service Compendium Specification R4, section 101 under "Log Service Specification." Recommended output per log level for CIShell algorithms are listed in table 5.1.

Log Level	**Recommended Use**
LOG_DEBUG	Used for problem determination and may be irrelevent to anyone but the algorithm developer.
LOG_ERROR	Indicates a problem occurred while the algorithm was executing. Indicators of possible fixes should be output to this level along with relevent information describing what went wrong, if possible.
LOG_INFO	Used for providing information about and while the algorithm is executing. It does not indicate a problem.
LOG_WARNING	Indicates that the algorithm will still be executed, but that outputs may not be what was expected. This is often in response to illogical, but still valid user inputs.

Table 5.1: Log Level Recommendations for Algorithms

Chapter 6

Preferences Service Specification

Version 1.1

6.1 Introduction

CIShell requires OSGi's standard `PreferencesService` to be installed in a CIShell-compliant system. This gives both the algorithms and the applications built on CIShell a standard preferences storage system to use. It is used mainly as a storage mechanism for data that is not necessarily adjustable by end-users. Things like, last opened file, recently opened files, or any sort of data that may need to be saved between sessions. This service has not been modified or extended. More information about the `PreferencesService` is available in the OSGi Service Platform Service Compendium Specification R4, section 106 under "Preferences Service Specification." Algorithm developers wishing to expose user-adjustable preferences, should refer to section 2.6 for more information on this separate task.

Chapter 7

Data Manager Application Service Specification

Version 1.0

7.1 Introduction

The Data Manager Service is an optional service designed for application developers to have a central `Data` management system. Algorithm developers may use this too if they wish, but it is not guaranteed to be available across all CIShell-compliant systems.

7.1.1 Entities

- *DataManagerService* - The service interface for `Data` management.
- *DataManagerListener* - The interface for listening to events generated by the `DataManagerService`.
- *DataManagerAdapter* - This abstract class is an adapter implementation of `DataManagerListener` with empty method implementations.
- *Data* - The interface used to pass data (other than input parameters) and its metadata between algorithms.

7.2 org.cishell.app.service.datamanager

Interface **DataManagerListener**

A listener that is notified of changes in the DataManagerService

Declaration

```
public interface DataManagerListener
```

- *dataAdded*
  `public void` **dataAdded**`( Data` **data**`, java.lang.String` **label** `)`
  - **Usage**
    - Notifies that a Data object has been added to the associated DataManagerService
  - **Parameters**
    - `data` - The added Data object
    - `label` - The label assigned to the Data object

- *dataLabelChanged*
  `public void` **dataLabelChanged**`( Data` **data**`, java.lang.String` **label** `)`
  - **Usage**
    - Notifies that a Data object has had its label changed
  - **Parameters**
    - `data` - The Data object
    - `label` - The new label

- *dataRemoved*
  `public void` **dataRemoved**`( Data` **data** `)`
  - **Usage**
    - Notifies that a Data object has been removed from the associated DataManagerService
  - **Parameters**
    - `data` - The removed Data object

- *dataSelected*
  `public void` **dataSelected**`( Data[]` **data** `)`
  - **Usage**
    - Notifies that a set of data objects have been selected in the associated DataManagerService
  - **Parameters**
    - `data` - The selected Data objects

INTERFACE **DataManagerService**

A service for managing loaded Data objects. DataManagerListeners may be registered to be notified of changes in the data manager.
Application developers are encouraged to use this service for managing the models they have loaded into memory. Algorithm developers are encouraged not to use this service as it is not guaranteed to be available like the standard CIShell services are.

Declaration

public interface DataManagerService

Methods

- *addData*
  `public void` **addData**`( Data` **data** `)`
  - **Usage**
    - Adds a Data object to the manager
  - **Parameters**
    - `data` - The data object

- *addDataManagerListener*
  `public void` **addDataManagerListener**`( DataManagerListener` **listener** `)`
  - **Usage**
    - Adds a DataManagerListener that will be notified as Data objects are added, removed, and selected
  - **Parameters**
    - `listener` - The listener to be notified of events

- *getAllData*
  `public Data` **getAllData**`( )`
  - **Usage**
    - Returns all of the Data objects loaded into the manager
  - **Returns** - An array of DataModels, length may be zero

- *getLabel*
  `public java.lang.String` **getLabel**`( Data` **data** `)`
  - **Usage**
    - Returns the label for a stored Data object
  - **Parameters**
    - `data` - The Data object
  - **Returns** - A label for the Data object

- *getSelectedData*
  `public Data` **getSelectedData**`( )`
  - **Usage**
    - Returns The Data objects that have been selected in the manager
  - **Returns** - An array of Data objects, length may be zero

---

- *removeData*
  `public void` **removeData**`( Data` **data** `)`
  - **Usage**
    - Removes a Data object from the manager
  - **Parameters**
    - `data` - The data object

- *removeDataManagerListener*
  `public void` **removeDataManagerListener**`( DataManagerListener` **listener** `)`
  - **Usage**
    - Removes the DataManagerListener from the listener group and will no longer notify it of events
  - **Parameters**
    - `listener` - The listener to be removed

- *setLabel*
  `public void` **setLabel**`( Data` **data**`, java.lang.String` **label** `)`
  - **Usage**
    - Set the label to be used for the Data object. The model manager is free to change the label so that it is unique.
  - **Parameters**
    - `data` - The Data
    - `label` - The new label for the data model

- *setSelectedData*
  `public void` **setSelectedData**`( Data[]` **data** `)`
  - **Usage**
    - Sets which Data objects are selected in the manager. If a given Data object in the array of Data objects is not in the data manager, then it will be automatically added before selection.
  - **Parameters**
    - `data` - The data objects to select

Class **DataManagerAdapter**

An abstract adapter class for notification of changes in the DataManagerService. The methods in this class are empty. This class exists as a convenience for creating listener objects.

Declaration

public class DataManagerAdapter
extends java.lang.Object
implements DataManagerListener

CONSTRUCTORS

- *DataManagerAdapter*
  `public` **DataManagerAdapter**`( )`

METHODS

- *dataAdded*
  `public void` **dataAdded**`( Data` **data**`, java.lang.String` **label** `)`
  - **See Also**
    * `org.cishell.app.service.datamanager.DataManagerListener.dataAdded( org.cishell.framework.data.Data, java.lang.String )`

- *dataLabelChanged*
  `public void` **dataLabelChanged**`( Data` **data**`, java.lang.String` **label** `)`
  - **See Also**
    * `org.cishell.app.service.datamanager.DataManagerListener.-dataLabelChanged( org.cishell.framework.data.Data, java.lang.String )`

- *dataRemoved*
  `public void` **dataRemoved**`( Data` **data** `)`
  - **See Also**
    * `org.cishell.app.service.datamanager.DataManagerListener.dataRemoved( org.cishell.framework.data.Data )` (in 7.2, page 49)

- *dataSelected*
  `public void` **dataSelected**`( Data[]` **data** `)`
  - **See Also**
    * `org.cishell.app.service.datamanager.DataManagerListener.dataSelected( org.cishell.framework.data.Data[] )` (in 7.2, page 49)

Chapter 8

Scheduler Application Service Specification

Version 1.0

8.1 Introduction

The Scheduler Service is an optional service designed for application developers to schedule algorithms for later monitoring and execution. Algorithm developers may use this too if they wish, but it is not guaranteed to be available across all CIShell-compliant systems.

8.1.1 Entities

- *SchedulerService* - The service interface for scheduling algorithms to be monitored and executed.
- *SchedulerListener* - The interface for listening to events generated by the `SchedulerService`.
- *SchedulerAdapter* - This abstract class is an adapter implementation of `SchedulerListener` with empty method implementations.

8.2 org.cishell.app.service.scheduler

Interface **SchedulerListener**

A listener that is notified of events happening in a SchedulerService

Declaration

```
public interface SchedulerListener
```

- *algorithmError*
  `public void` **algorithmError**`( Algorithm` **algorithm**`, java.lang.Throwable` **error** `)`
  - **Usage**
    - Notification that an Algorithm had an error while being executed
  - **Parameters**
    - `algorithm` - The scheduled Algorithm
    - `error` - The error it threw while executing

- *algorithmFinished*
  `public void` **algorithmFinished**`( Algorithm` **algorithm**`, Data[]` **createdData** `)`
  - **Usage**
    - Notification that an Algorithm has finished executing
  - **Parameters**
    - `algorithm` - The scheduled Algorithm
    - `createdData` - The Data array it returned, or `null` if it returned `null`

- *algorithmRescheduled*
  `public void` **algorithmRescheduled**`( Algorithm` **algorithm**`, java.util.Calendar` **time** `)`
  - **Usage**
    - Notification that an already scheduled Algorithm has been rescheduled to be run at a different time
  - **Parameters**
    - `algorithm` - The scheduled Algorithm
    - `time` - The new time the Algorithm is scheduled to be run

---

- *algorithmScheduled*
  `public void` **algorithmScheduled**`( Algorithm` **algorithm**`, java.util.Calendar` **time** `)`
  - **Usage**
    - Notification that an Algorithm has been scheduled to be run at a certain time
  - **Parameters**
    - `algorithm` - The scheduled Algorithm
    - `time` - The time is scheduled to be run

- *algorithmStarted*
  `public void` **algorithmStarted**`( Algorithm` **algorithm** `)`
  - **Usage**
    - Notification that an Algorithm has started execution
  - **Parameters**

        * `algorithm` - The scheduled algorithm

- *algorithmUnscheduled*
  `public void` **algorithmUnscheduled**`( Algorithm` **algorithm** `)`
  - **Usage**
    * Notification that an already scheduled Algorithm has been unscheduled and will therefore not be run
  - **Parameters**
    * `algorithm` - The scheduled Algorithm that was unscheduled

- *schedulerCleared*
  `public void` **schedulerCleared**`( )`
  - **Usage**
    * Notification that the scheduler's schedule of Algorithms to be run has been cleared

---

- *schedulerRunStateChanged*
  `public void` **schedulerRunStateChanged**`( boolean` **isRunning** `)`
  - **Usage**
    * Notification that the scheduler's run state (paused or unpaused) has changed
  - **Parameters**
    * `isRunning` - `true` if it is now running, `false` if it is no longer running (paused)

Interface **SchedulerService**

A service for scheduling Algorithms to be run. SchedulerListeners may be registered to be notified of events.
Application Developers are encouraged to use this service for scheduling Algorithms to be run. Algorithm developers are encouraged not to use this service as it is not guaranteed to be available like the standard CIShell services are.

Declaration

```
public interface SchedulerService
```

Methods

- *addSchedulerListener*
  `public void` **addSchedulerListener**`( SchedulerListener` **listener** `)`
  - **Usage**
    * Adds a listener to be notified of events happening in the scheduler
  - **Parameters**

        * `listener` - The listener to be added

- *clearSchedule*
  `public void` **clearSchedule**`( )`
  - **Usage**
    * Clears all currently scheduled Algorithms to be run. If an Algorithm is already running, then it will continue to run until finished.

- *getScheduledAlgorithms*
  `public Algorithm` **getScheduledAlgorithms**`( )`
  - **Usage**
    * Returns an array of Algorithms that the scheduler has scheduled. This includes the Algorithms that are currently running and the ones queued to be run. This also just gives a snapshot of the current set of scheduled Algorithms, so it is not guaranteed to be accurate even directly after the method returns.
  - **Returns** - The set of Algorithms currently scheduled in the scheduler

---

- *getScheduledTime*
  `public java.util.Calendar` **getScheduledTime**`( Algorithm` **algorithm** `)`
  - **Usage**
    * Returns the time in which a scheduled Algorithm is scheduled to be run. The time may be in the past if the Algorithm is already running or `null` if the Algorithm is not scheduled.
  - **Parameters**
    * `algorithm` - The Algorithm
  - **Returns** - The scheduled time for the Algorithm to run or `null` if the Algorithm is not scheduled or has completed execution

---

- *getServiceReference*
  `public org.osgi.framework.ServiceReference` **getServiceReference**`( Algorithm` **algorithm** `)`
  - **Usage**
    * Returns an Algorithm's associated ServiceReference if one was provided when the Algorithm was scheduled
  - **Parameters**
    * `algorithm` - The Algorithm
  - **Returns** - Its associated ServiceReference

- *isEmpty*
  `public boolean` **isEmpty**`( )`
  - **Usage**
    * Returns if there are any Algorithms scheduled
  - **Returns** - Whether there are any Algorithms scheduled

- *isRunning*
  `public boolean` **isRunning**`( )`
  - **Usage**
    - Returns whether the scheduler is running
  - **Returns** - if the scheduler is running

- *removeSchedulerListener*
  `public void` **removeSchedulerListener**`( SchedulerListener` **listener** `)`
  - **Usage**
    - Removes a SchedulerListener from the group of listeners listening for scheduler events. This method has no effect if the listener isn't in the group of listeners.
  - **Parameters**
    - `listener` - The listener to be removed

- *reschedule*
  `public boolean` **reschedule**`( Algorithm` **algorithm**`, java.util.Calendar` **newTime** `)`
  - **Usage**
    - Reschedules an already scheduled Algorithm to be run at a different time. If the Algorithm is not scheduled already, then this method will have no effect and will return `false`.
  - **Parameters**
    - `algorithm` - The Algorithm already scheduled
    - `newTime` - The revised time in which to run the Algorithm
  - **Returns** - If the Algorithm was successfully rescheduled

---

- *runNow*
  `public void` **runNow**`( Algorithm` **algorithm**`,`
  `org.osgi.framework.ServiceReference` **ref** `)`
  - **Usage**
    - Schedules an Algorithm to be run immediately. If there are simply not enough resources to run it, it will still have to wait until there are enough resources to fulfill the request.
  - **Parameters**
    - `algorithm` - The algorithm to be run
    - `ref` - A reference to the Algorithm's associated service, may be `null`

- *schedule*
  `public void` **schedule**`( Algorithm` **algorithm**`,`
  `org.osgi.framework.ServiceReference` **ref** `)`
  - **Usage**
    - Schedules an Algorithm to be run when convenient. This schedules an Algorithm to be run now, but gives no urgent priority to it. Most Algorithms will be scheduled in this way.

- **Parameters**
  - `algorithm` - The Algorithm to be scheduled
  - `ref` - A reference to the Algorithm's associated service, may be `null`

- *schedule*
  `public void` **schedule**`( Algorithm` **algorithm**`,`
  `org.osgi.framework.ServiceReference` **ref**`, java.util.Calendar` **time** `)`
  - **Usage**
    - Schedules an Algorithm to be run at a specific time. The Algorithm will be run at the given time unless there is simply not enough resources at that time. In which case it would wait until there are enough resources to fulfill the request.
  - **Parameters**
    - `algorithm` - The Algorithm to be scheduled
    - `ref` - A reference to the Algorithm's associated service, may be `null`
    - `time` - What time this Algorithm should be run

- *setRunning*
  `public void` **setRunning**`( boolean` **isRunning** `)`
  - **Usage**
    - Pauses or unpauses the running of new Algorithms in the scheduler
  - **Parameters**
    - `isRunning` - `true` to pause, `false` to unpause

- *unschedule*
  `public boolean` **unschedule**`( Algorithm` **algorithm** `)`
  - **Usage**
    - Unschedules an already scheduled, but not running Algorithm from the scheduler. Tries to unschedule an Algorithm from the scheduler. If the Algorithm isn't in the scheduler or if the Algorithm is already running then this method returns `false`.
  - **Parameters**
    - `algorithm` - The Algorithm to remove from the scheduler
  - **Returns** - If the Algorithm was successfully unscheduled

Class **SchedulerAdapter**

An abstract adapter class for notification of events happening in a SchedulerService. The methods in this class are empty. This class exists as a convenience for creating listener objects.

Declaration

public abstract class SchedulerAdapter **extends** java.lang.Object **implements** SchedulerListener

CONSTRUCTORS

- *SchedulerAdapter*
  `public` **SchedulerAdapter**`( )`

METHODS

- *algorithmError*
  `public void` **algorithmError**`( Algorithm` **algorithm**`, java.lang.Throwable` **error** `)`

- *algorithmFinished*
  `public void` **algorithmFinished**`( Algorithm` **algorithm**`, Data[]` **createdData** `)`
- *algorithmRescheduled*
  `public void` **algorithmRescheduled**`( Algorithm` **algorithm**`, java.util.Calendar` **time** `)`
- *algorithmScheduled*
  `public void` **algorithmScheduled**`( Algorithm` **algorithm**`, java.util.Calendar` **time** `)`
- *algorithmStarted*
  `public void` **algorithmStarted**`( Algorithm` **algorithm** `)`
- *algorithmUnscheduled*
  `public void` **algorithmUnscheduled**`( Algorithm` **algorithm** `)`
- *schedulerCleared*
  `public void` **schedulerCleared**`( )`
- *schedulerRunStateChanged*
  `public void` **schedulerRunStateChanged**`( boolean` **isRunning** `)`

Appendix A

Apache 2.0 License

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You

institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.