Topic

Topic

Topic

- ☐ Code Reviewers
  - ☐ Testers
    - ☐ Karla
    - ☐ Kate
    - ☐ Chris
  - ☐ Commandos
    - ☐ Ilana Dashevky
    - ☐ Perry Rose
    - ☐ Marsha Allen
  - ☐ Developers
    - ☐ Andy
    - ☐ Tom
    - ☐ Pat
    - ☐ Gary - did not attend

APT Developer Primer

The following are some concepts and constructs used extensively in APT development that are not being reviewed, but should be understood in the context of the code review.

Java Constructs

Enums

Enums are a Java construct that allows explicitly named enumerated instances of a particular concept. They are often used in place of static int collections often seen in other programming languages. A Java Enum is a light-weight class, so it is fast to use, but also provides all the functionality of a class.

For more on Enums, there is a short description available here:

http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html

Generics

Generics are a means of identifying the underlying classes contained by another class, and ensuring type safety at compile time. It is an alternative to explicit class casting done at run time, which is necessary to have access to class methods of the underlying object. In the case of Collections, this is a more object-oriented approach to providing the same functionality of a typed array.

With generics you *know* the class of the object you are working with at compile time. With casting, you *hope* you know the class of the object you are working with at run time.

For more on Generics, there is a short description available here:

http://java.sun.com/j2se/1.5.0/docs/guide/language/generics.html

APT Team Libraries

CoSI

The CoSI (Constraint Sequencing infrastructure) is a library of objects, and an infrastructure to support automatic rerunning of methods when dependent variables change. The basic design is that there are Properties, Constraints, and a Propagator.

The Propagator runs the Constraints. It has a queue of ones that are ready to run, and runs them sequentially. A Constraint is the equivalent of a method. It accesses some Properties, and sets others, all within the context of a single run() method. While the Constraint is running, all the Properties that it accesses track that this Constraint is dependent on that Property. When the Property changes, it tells the Propagator to add all dependent Constraints to its queue. This is a different way of handling property change events than the one normally supported by Java, but it allows a more declarative expression of the logic. This design was based on that used by Trans.

Tina

Tina (Tina is no acronym) is a library of objects designed to support the proposal hierarchy. It is the basis for the HST and JWST document models. It also provides the GUI elements that are used for editing proposals.

The document model is divided into Elements and Fields. A TinaDocumentElement is a complex type that can be inserted in a document hierarchy in a parent child relationship with the document as a whole. TinaFields are more akin to attributes of a TinaDocumentElement. They represent a specific value and present a specific way of editing that one value. Tina provides all the infrastructure for managing the relationship between the Elements.

For the JWST project, we have begun to merge the Cosi Properties into the Tina Fields, allowing Tina Fields to be used to force Constraints to rerun.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.stsci.edu/JWST/APT/Instrument/Nircam"
    targetNamespace="http://www.stsci.edu/JWST/APT/Instrument/Nircam"
    elementFormDefault="qualified"
    version="1">

    <xs:simpleType name="ObjectTypeType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="FAINT"/>
            <xs:enumeration value="BRIGHT"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="ModuleType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="A"/>
            <xs:enumeration value="B"/>
            <xs:enumeration value="ALL"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="ReadoutRegionType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="FULL"/>
            <xs:enumeration value="SUB16"/>
            <xs:enumeration value="SUB48"/>
            <xs:enumeration value="SUB96"/>
            <xs:enumeration value="SUB320"/>
            <xs:enumeration value="SUB640"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="DitherType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="tbd1"/>
            <xs:enumeration value="tbd2"/>
            <xs:enumeration value="tbd3"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="FilterType">
        <xs:restriction base="xs:string">
            <!-- Single Short Filters -->
            <xs:enumeration value="F070W"/>
            <xs:enumeration value="F090W"/>
            <xs:enumeration value="F115W"/>
            <xs:enumeration value="F150W"/>
            <xs:enumeration value="F150W2"/>
            <xs:enumeration value="F200W"/>
```

```
        <xs:enumeration value="F140M"/>
        <xs:enumeration value="F182M"/>
        <xs:enumeration value="F210M"/>
        <xs:enumeration value="F187N"/>
        <xs:enumeration value="F212N"/>
        <xs:enumeration value="WL3"/>
        <!-- Crossed Short Filters -->
        <xs:enumeration value="F162M_F150W2"/>
        <xs:enumeration value="F164N_F150W"/>
        <xs:enumeration value="F164N_F150W2"/>
        <xs:enumeration value="F225N_F150W2"/>
        <!-- Single Long Filters -->
        <xs:enumeration value="F277W"/>
        <xs:enumeration value="F322W2"/>
        <xs:enumeration value="F356W"/>
        <xs:enumeration value="F444W"/>
        <xs:enumeration value="F250M"/>
        <xs:enumeration value="F300M"/>
        <xs:enumeration value="F335M"/>
        <xs:enumeration value="F360M"/>
        <xs:enumeration value="F410M"/>
        <xs:enumeration value="F430M"/>
        <xs:enumeration value="F460M"/>
        <xs:enumeration value="F480M"/>
        <!-- Crossed Long Filters -->
        <xs:enumeration value="F323N+F356W"/>
        <xs:enumeration value="F323N+F322W2"/>
        <xs:enumeration value="F405N+F444W"/>
        <xs:enumeration value="F405N+F410M"/>
        <xs:enumeration value="F418N+F444W"/>
        <xs:enumeration value="F418N+F410M"/>
        <xs:enumeration value="F466N+F444W"/>
        <xs:enumeration value="F466N+F460M"/>
        <xs:enumeration value="F470N+F444W"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="ReadoutPatternType">
    <xs:restriction base="xs:string">
        <xs:enumeration value="DEEP8"/>
        <xs:enumeration value="DEEP2"/>
        <xs:enumeration value="MEDIUM8"/>
        <xs:enumeration value="MEDIUM2"/>
        <xs:enumeration value="SHALLOW4"/>
        <xs:enumeration value="SHALLOW2"/>
        <xs:enumeration value="BRIGHT2"/>
        <xs:enumeration value="BRIGHT1"/>
        <xs:enumeration value="RAPID"/>
    </xs:restriction>
</xs:simpleType>
```

```
    <xs:simpleType name="MechanismType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Filter"/>
            <xs:enumeration value="Pupil"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="WheelType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="SHORTA"/>
            <xs:enumeration value="SHORTB"/>
            <xs:enumeration value="LONGA"/>
            <xs:enumeration value="LONGB"/>
            <xs:enumeration value="ALL"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="PupilType">
        <xs:restriction base="xs:string">
            <!-- BOTH -->
            <xs:enumeration value="CLEAR"/>
            <xs:enumeration value="FLAT"/>
            <xs:enumeration value="CORON1"/>
            <xs:enumeration value="CORON2"/>
            <xs:enumeration value="PINHOLES"/>
            <!-- Long Pupil Types -->
            <xs:enumeration value="F323N"/>
            <xs:enumeration value="F418N"/>
            <xs:enumeration value="F470N"/>
            <xs:enumeration value="F405N"/>
            <xs:enumeration value="F466N"/>
            <xs:enumeration value="GRISM1"/>
            <xs:enumeration value="GRISM2"/>
            <!-- Short Pupil Types -->
            <xs:enumeration value="F164N"/>
            <xs:enumeration value="F162M"/>
            <xs:enumeration value="GDHS1"/>
            <xs:enumeration value="WL1"/>
            <xs:enumeration value="WL2"/>
            <xs:enumeration value="F225N"/>
            <xs:enumeration value="GDHS2"/>
        </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

```
<jxb:bindings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/jaxb/bindingschema_2_0.xsd"
    xmlns:jxb="http://java.sun.com/xml/ns/jaxb" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    version="1.0" schemaLocation="Nircam.xsd">

    <jxb:schemaBindings>
        <jxb:nameXmlTransform>
            <jxb:elementName prefix="Jaxb"/>
            <jxb:typeName prefix="Jaxb"/>
        </jxb:nameXmlTransform>
    </jxb:schemaBindings>
</jxb:bindings>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns="http://www.stsci.edu/JWST/APT/Template/NircamCoron"
    xmlns:nircam="http://www.stsci.edu/JWST/APT/Instrument/Nircam"
    targetNamespace="http://www.stsci.edu/JWST/APT/Template/NircamCoron"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    version="3">

    <xs:import namespace="http://www.stsci.edu/JWST/APT/Instrument/Nircam" schemaLocation="../../InstrumentSchemas/Nircam.xsd"/>

    <xs:element name="NircamCoron">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ObjectType" type="nircam:ObjectTypeType" minOccurs="0"/>
                <xs:element name="CoronMask" type="CoronMaskType" minOccurs="0"/>
                <xs:element name="AcqTargetID" type="xs:string" minOccurs="0"/>
                <xs:element name="AcqFilter" type="nircam:FilterType" minOccurs="0"/>
                <xs:element name="AcqReadoutPattern" type="nircam:ReadoutPatternType" minOccurs="0"/>
                <xs:element name="AcqGroups" type="xs:int" minOccurs="0"/>
                <xs:element name="Filters" type="FiltersType" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="FiltersType">
        <xs:sequence>
            <xs:element name="FilterConfig" type="FilterConfigType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="FilterConfigType">
        <xs:sequence>
            <xs:element name="Filter" type="nircam:FilterType" minOccurs="0"/>
            <xs:element name="RequestedExposureTime" type="xs:double" minOccurs="0"/>
            <xs:element name="ReadoutPattern" type="nircam:ReadoutPatternType" minOccurs="0"/>
            <xs:element name="NumberOfGroups" type="xs:int" minOccurs="0"/>
            <xs:element name="NumberOfIntegrations" type="xs:int" minOccurs="0"/>
            <xs:element name="CalculatedExposureTime" type="xs:double" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>

    <xs:simpleType name="CoronMaskType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="MASK210R"/>
            <xs:enumeration value="MASKSWB"/>
            <xs:enumeration value="MASK335R"/>
            <xs:enumeration value="MASK430R"/>
            <xs:enumeration value="MASKLWB"/>
        </xs:restriction>
```

```
    </xs:simpleType>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns="http://www.stsci.edu/JWST/APT/Template/NircamDark"
    xmlns:nircam="http://www.stsci.edu/JWST/APT/Instrument/Nircam"
    targetNamespace="http://www.stsci.edu/JWST/APT/Template/NircamDark"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    version="2">

    <xs:import namespace="http://www.stsci.edu/JWST/APT/Instrument/Nircam" schemaLocation="../../InstrumentSchemas/Nircam.xsd"/>

    <xs:element name="NircamDark">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Module" type="nircam:ModuleType" minOccurs="0"/>
                <xs:element name="ReadoutRegion" type="nircam:ReadoutRegionType" minOccurs="0"/>
                <xs:element name="Exposures" type="ExposuresType" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="ExposuresType">
        <xs:sequence>
            <xs:element name="ExposureConfig" type="ExposureConfigType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="ExposureConfigType">
        <xs:sequence>
            <xs:element name="ReadoutPattern" type="nircam:ReadoutPatternType" minOccurs="0"/>
            <xs:element name="Exposures" type="xs:int" minOccurs="0"/>
            <xs:element name="Groups" type="xs:int" minOccurs="0"/>
            <xs:element name="Integrations" type="xs:int" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns="http://www.stsci.edu/JWST/APT/Template/NircamFocus"
    xmlns:nircam="http://www.stsci.edu/JWST/APT/Instrument/Nircam"
    targetNamespace="http://www.stsci.edu/JWST/APT/Template/NircamFocus"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    version="3">

    <xs:import namespace="http://www.stsci.edu/JWST/APT/Instrument/Nircam" schemaLocation="../../InstrumentSchemas/Nircam.xsd"/>

    <xs:element name="NircamFocus">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Module" type="nircam:ModuleType" minOccurs="0"/>
                <xs:element name="LongFilter" type="nircam:FilterType" minOccurs="0"/>
                <xs:element name="ShortFilter" type="nircam:FilterType" minOccurs="0"/>
                <xs:element name="LongPupil" type="nircam:PupilType" minOccurs="0"/>
                <xs:element name="ShortPupil" type="nircam:PupilType" minOccurs="0"/>
                <xs:element name="Groups" type="xs:int" minOccurs="0"/>
                <xs:element name="Integrations" type="xs:int" minOccurs="0"/>
                <xs:element name="ReadoutPattern" type="nircam:ReadoutPatternType" minOccurs="0"/>

                <xs:element name="StartingPositionSteps" type="LinearActuatorType" minOccurs="0"/>
                <xs:element name="StartingPositionSensorUnits" type="LinearActuatorType" minOccurs="0"/>
                <xs:element name="StartingMotorPhase" type="LinearActuatorType" minOccurs="0"/>
                <xs:element name="Positions" type="PositionsType" minOccurs="0"/>
                <xs:element name="ReturnToStart" type="xs:boolean" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="LinearActuatorType">
        <xs:sequence>
            <xs:element name="Value1" type="xs:int" minOccurs="0"/>
            <xs:element name="Value2" type="xs:int" minOccurs="0"/>
            <xs:element name="Value3" type="xs:int" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="PositionsType">
        <xs:sequence>
            <xs:element name="Position" type="LinearActuatorType" minOccurs="0" maxOccurs="10"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns="http://www.stsci.edu/JWST/APT/Template/NircamImaging"
    xmlns:nircam="http://www.stsci.edu/JWST/APT/Instrument/Nircam"
    targetNamespace="http://www.stsci.edu/JWST/APT/Template/NircamImaging"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    version="1">

    <xs:import namespace="http://www.stsci.edu/JWST/APT/Instrument/Nircam" schemaLocation="../../InstrumentSchemas/Nircam.xsd"/>

    <xs:element name="NircamImaging">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ObjectType" type="nircam:ObjectTypeType" minOccurs="0"/>
                <xs:element name="Module" type="nircam:ModuleType" minOccurs="0"/>
                <xs:element name="ReadoutRegion" type="nircam:ReadoutRegionType" minOccurs="0"/>
                <xs:element name="Filters" type="FiltersType" minOccurs="0"/>
                <xs:element name="Dither" type="nircam:DitherType" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="FiltersType">
        <xs:sequence>
            <xs:element name="FilterConfig" type="FilterConfigType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="FilterConfigType">
        <xs:sequence>
            <xs:element name="ShortFilter" type="nircam:FilterType" minOccurs="0"/>
            <xs:element name="LongFilter" type="nircam:FilterType" minOccurs="0"/>
            <xs:element name="RequestedExposureTime" type="xs:double" minOccurs="0"/>
            <xs:element name="ReadoutPattern" type="nircam:ReadoutPatternType" minOccurs="0"/>
            <xs:element name="Groups" type="xs:int" minOccurs="0"/>
            <xs:element name="Integrations" type="xs:int" minOccurs="0"/>
            <xs:element name="ActualExposureTime" type="xs:double" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns="http://www.stsci.edu/JWST/APT/Template/NircamInternalFlat"
    xmlns:nircam="http://www.stsci.edu/JWST/APT/Instrument/Nircam"
    targetNamespace="http://www.stsci.edu/JWST/APT/Template/NircamInternalFlat"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    version="2">

    <xs:import namespace="http://www.stsci.edu/JWST/APT/Instrument/Nircam" schemaLocation="../../InstrumentSchemas/Nircam.xsd"/>

    <xs:element name="NircamInternalFlat">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Module" type="nircam:ModuleType" minOccurs="0"/>
                <xs:element name="Filters" type="FiltersType" minOccurs="0"/>
                <xs:element name="Exposures" type="xs:int" minOccurs="0"/>
                <xs:element name="Integrations" type="xs:int" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="FiltersType">
        <xs:sequence>
            <xs:element name="FilterConfig" type="FilterConfigType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="FilterConfigType">
        <xs:sequence>
            <xs:element name="ShortFilter" type="nircam:FilterType" minOccurs="0"/>
            <xs:element name="LongFilter" type="nircam:FilterType" minOccurs="0"/>
            <xs:element name="Exposures" type="xs:int" minOccurs="0"/>
            <xs:element name="Integrations" type="xs:int" minOccurs="0"/>
            <xs:element name="ActualExposureTime" type="xs:double" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns="http://www.stsci.edu/JWST/APT/Template/NircamWheelExercise"
    xmlns:nircam="http://www.stsci.edu/JWST/APT/Instrument/Nircam"
    targetNamespace="http://www.stsci.edu/JWST/APT/Template/NircamWheelExercise"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    version="2">

    <xs:import namespace="http://www.stsci.edu/JWST/APT/Instrument/Nircam" schemaLocation="../../InstrumentSchemas/Nircam.xsd"/>

    <xs:element name="NircamWheelExercise">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Mechanism" type="nircam:MechanismType" minOccurs="0"/>
                <xs:element name="Wheel" type="nircam:WheelType" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="Rotations" type="xs:int" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```

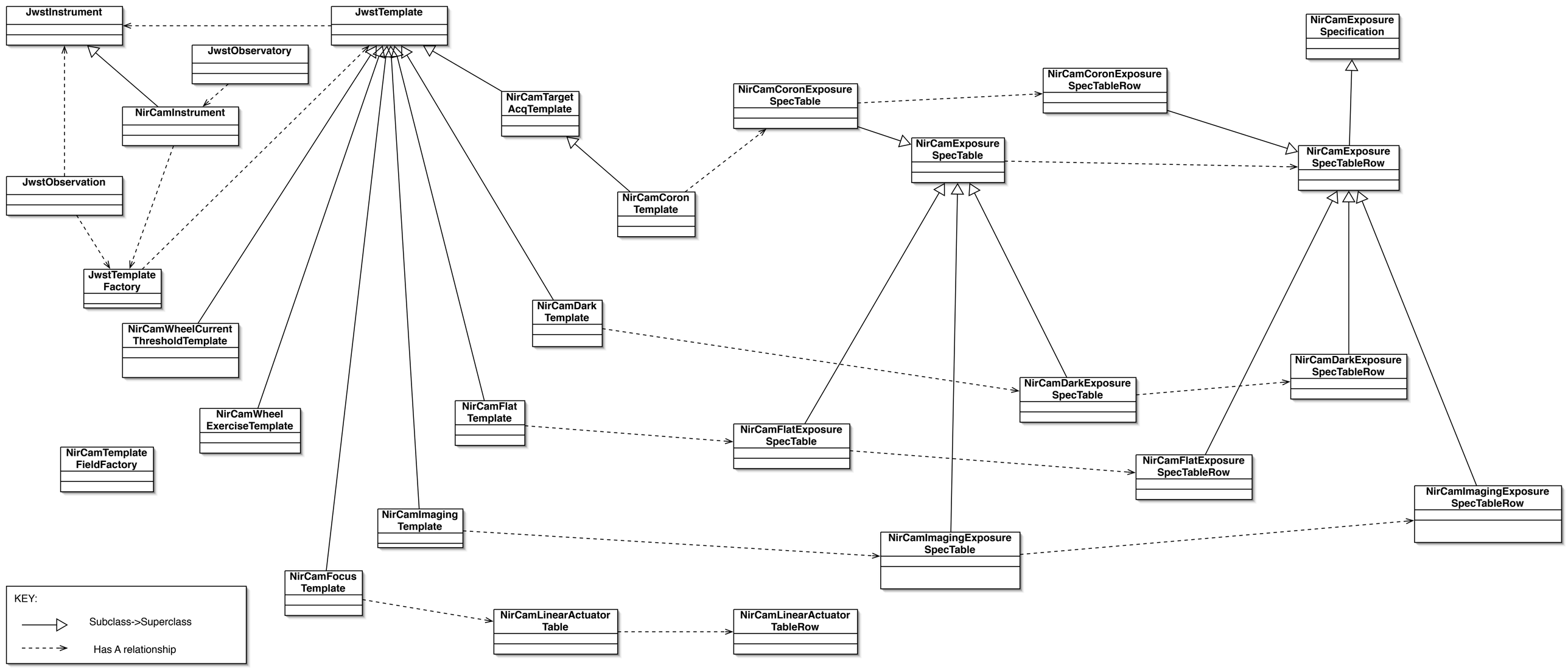
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns="http://www.stsci.edu/JWST/APT/Template/NircamWheelThresholdCurrent"
    xmlns:nircam="http://www.stsci.edu/JWST/APT/Instrument/Nircam"
    targetNamespace="http://www.stsci.edu/JWST/APT/Template/NircamWheelThresholdCurrent"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    version="1">

    <xs:import namespace="http://www.stsci.edu/JWST/APT/Instrument/Nircam" schemaLocation="../../InstrumentSchemas/Nircam.xsd"/>

    <xs:element name="NircamWheelThresholdCurrent">
        <xs:complexType>
            <xs:sequence>

            </xs:sequence>
        </xs:complexType>
    </xs:element>

</xs:schema>
```


JwstInstrument
JwstTemplate
JwstObservatory
NirCamInstrument
NirCamTarget AcqTemplate
NirCamCoronExposure SpecTable
NirCamCoronExposure SpecTableRow
NirCamExposure Specification
NirCamExposure SpecTable
NirCamExposure SpecTableRow
JwstObservation
NirCamCoron Template
JwstTemplate Factory
NirCamWheelCurrent ThresholdTemplate
NirCamDark Template
NirCamDarkExposure SpecTable
NirCamDarkExposure SpecTableRow
NirCamWheel ExerciseTemplate
NirCamFlat Template
NirCamFlatExposure SpecTable
NirCamFlatExposure SpecTableRow
NirCamTemplate FieldFactory
NirCamImaging Template
NirCamImagingExposure SpecTable
NirCamImagingExposure SpecTableRow
NirCamFocus Template
NirCamLinearActuator Table
NirCamLinearActuator TableRow
KEY:
Subclass->Superclass
Has A relationship

```java
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.instrument;

import java.util.Arrays;

/**
 * JwstObservatory - represents the actual telescope, contains the instruments and any
 * parameters we need to know about the Observatory itself.
 *
 * @author Rob Douglas
 */
public class JwstObservatory {
    //--------------------Statics--------------------
    private static final List<JwstInstrument> instruments = Arrays.asList(new JwstInstrument[]{
            MiriInstrument.getInstance(),
            NirCamInstrument.getInstance(),
            NirSpecInstrument.getInstance(),
            TfiInstrument.getInstance()
    });

    //--------------------Constructors--------------------
    private JwstObservatory (){}

    //--------------------Static Accessors--------------------
    public static List<JwstInstrument> getInstruments () {
        return instruments;
    }
}
```

```java
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.instrument;

import java.util.List;
/**
 * JwstInstrument - Defines what must be true about all instruments in JWST.
 *
 * @author Rob Douglas
 */
public abstract class JwstInstrument {

    private final String name;

    //--------------------Constructors--------------------

    public JwstInstrument(String iName) {
        name = iName;
    }

    //--------------------Accessors--------------------
    public String getName() {
        return name;
    }

    public abstract List<JwstFilter> getAcqFilters();

    //--------------------Methods--------------------

    public abstract double computeOverheadTime ();

    @Override
    public String toString() {
        return getName();
    }

    //--------------------Enums--------------------
    public enum JwstFilter {}
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.instrument;

import java.util.Arrays;

/**
 * NirCamInstrument - represents the NIRCam and all its associated data parameter values that
 * can be set for NIRCam.
 *
 * @author Rob Douglas
 */
public class NirCamInstrument extends JwstInstrument {
    //--------------------Statics--------------------
    private static final NirCamInstrument NIRCAM_INSTRUMENT = new NirCamInstrument();

    public static final NirCamFilter[] ACQ_FILTERS = new NirCamFilter[] {
        NirCamFilter.F182M,
        NirCamFilter.F335M
    };

    //--------------------Constructors--------------------
    //Singleton
    private NirCamInstrument (){
        super("NIRCAM");
    }

    //--------------------Static Accessor--------------------
    public static NirCamInstrument getInstance (){
        return NIRCAM_INSTRUMENT;
    }

    //--------------------Accessors--------------------
    @Override
    public List getAcqFilters() {
        return Arrays.asList(ACQ_FILTERS);
    }

    //--------------------Methods--------------------
    @Override
    public double computeOverheadTime() {
        // TODO Auto-generated method stub
        return 0;
    }

    //--------------------Enums--------------------
    //Module
    public enum NirCamModule {
        A,
        B,
        ALL
```

```
}

//Subarray
public enum NirCamSubarray {
    FULL(2048,2048),
    SUB16(16,16),
    SUB48(48,48),
    SUB64(64,64),
    SUB96(96,96),
    SUB128(128,128),
    SUB320(320,320),
    SUB640(640,640);

    //--------------------Fields--------------------
    int rows;
    int columns;

    //--------------------Constructors--------------------
    NirCamSubarray (int iRows, int iColumns) {
        rows = iRows;
        columns = iColumns;
    }

    //--------------------Accessors--------------------

    public int getRows() {
        return rows;
    }

    public int getColumns() {
        return columns;
    }
}

//Wavelength Channels
public enum NirCamWavelength {
    SHORT,
    LONG,
    BOTH
}

//Filters
public enum NirCamFilter {
    //Short Wavelength Filters
    F070W("F070W", NirCamWavelength.SHORT),
    F090W("F090W", NirCamWavelength.SHORT),
    F115W("F115W", NirCamWavelength.SHORT),
    F150W("F150W", NirCamWavelength.SHORT),
    F150W2("F150W2", NirCamWavelength.SHORT),
    WL3("WL3", NirCamWavelength.SHORT),
```

```
F200W("F200W", NirCamWavelength.SHORT),
F140M("F140M", NirCamWavelength.SHORT),
F182M("F182M", NirCamWavelength.SHORT),
F210M("F210M", NirCamWavelength.SHORT),
F187N("F187N", NirCamWavelength.SHORT),
F212N("F212N", NirCamWavelength.SHORT),
F162M_F150W2("F162M+F150W2", NirCamWavelength.SHORT),
F164N_F150W("F164N+F150W", NirCamWavelength.SHORT),
F164N_F150W2("F164N+F150W2", NirCamWavelength.SHORT),
F225N_F150W2("F225N+F150W2", NirCamWavelength.SHORT),
//Long Wavelength Filters
F277W("F277W", NirCamWavelength.LONG),
F322W2("F322W2", NirCamWavelength.LONG),
F356W("F356W", NirCamWavelength.LONG),
F444W("F444W", NirCamWavelength.LONG),
F250M("F250M", NirCamWavelength.LONG),
F300M("F300M", NirCamWavelength.LONG),
F335M("F335M", NirCamWavelength.LONG),
F360M("F360M", NirCamWavelength.LONG),
F410M("F410M", NirCamWavelength.LONG),
F430M("F430M", NirCamWavelength.LONG),
F460M("F460M", NirCamWavelength.LONG),
F480M("F480M", NirCamWavelength.LONG),
F323N_F356W("F323N+F356W", NirCamWavelength.LONG),
F323N_F322W2("F323N+F322W2", NirCamWavelength.LONG),
F405N_F444W("F405N+F444W", NirCamWavelength.LONG),
F405N_F410M("F405N+F410M", NirCamWavelength.LONG),
F418N_F444W("F418N+F444W", NirCamWavelength.LONG),
F418N_F410M("F418N+F410M", NirCamWavelength.LONG),
F466N_F444W("F466N+F444W", NirCamWavelength.LONG),
F466N_F460M("F466N+F460M", NirCamWavelength.LONG),
F470N_F444W("F470N+F444W", NirCamWavelength.LONG);

//--------------------Fields--------------------
private final String name;
private final NirCamWavelength wavelength;

//--------------------Constructors--------------------
NirCamFilter (String iName, NirCamWavelength iWave) {
    name = iName;
    wavelength = iWave;
}

//--------------------Methods--------------------
/*
 * Returns a list of Filters that match the wavelength desired.  If null
 * is passed in, then all filters are returned.
 */
public static List<NirCamFilter> getFilters (NirCamWavelength iWavelength) {
    List<NirCamFilter> lFilters = new Vector<NirCamFilter>();
```

```
        for (int i = 0; i < values().length; i++) {
            NirCamFilter lFilter = values()[i];
            if (iWavelength==null || iWavelength == lFilter.wavelength) {
                lFilters.add(lFilter);
            }
        }
        return lFilters;
    }

    @Override
    public String toString() {
        return name;
    }
}

//Pupils
public enum NirCamPupil {
    //These are naturally ordered in alphabetical order.
    CLEAR(NirCamWavelength.BOTH),
    CORON1(NirCamWavelength.BOTH),
    CORON2(NirCamWavelength.BOTH),
    F162M(NirCamWavelength.SHORT),
    F164N(NirCamWavelength.SHORT),
    F225N(NirCamWavelength.SHORT),
    F323N(NirCamWavelength.LONG),
    F405N(NirCamWavelength.LONG),
    F418N(NirCamWavelength.LONG),
    F466N(NirCamWavelength.LONG),
    F470N(NirCamWavelength.LONG),
    FLAT(NirCamWavelength.BOTH),
    GDHS1(NirCamWavelength.SHORT),
    GDHS2(NirCamWavelength.SHORT),
    GRISM1(NirCamWavelength.LONG),
    GRISM2(NirCamWavelength.LONG),
    PINHOLES(NirCamWavelength.BOTH),
    WL1(NirCamWavelength.SHORT),
    WL2(NirCamWavelength.SHORT);

    //--------------------Fields--------------------
    private final NirCamWavelength wavelength;

    //--------------------Constructors--------------------
    NirCamPupil (NirCamWavelength iWave) {
        wavelength = iWave;
    }

    //--------------------Methods--------------------
    /*
     * Returns a list of Pupils that match the wavelength desired.  If null
     * is passed in, then all pupils are returned.
```

```
     */
    public static List<NirCamPupil> getPupils (NirCamWavelength iWavelength) {
        List<NirCamPupil> lPupils = new Vector<NirCamPupil>();
        for (int i = 0; i < values().length; i++) {
            NirCamPupil lPupil = values()[i];
            if (iWavelength==null || lPupil.wavelength==NirCamWavelength.BOTH ||
                    iWavelength == lPupil.wavelength) {
                lPupils.add(lPupil);
            }
        }
        return lPupils;
    }
}

//Readout Pattern
public enum NirCamReadPattern {
    DEEP8,
    DEEP2,
    MEDIUM8,
    MEDIUM2,
    SHALLOW4,
    SHALLOW2,
    BRIGHT2,
    BRIGHT1,
    RAPID
}

//Coronagraphic Masks
//Maps Enum of Masks to the Enum of Legal Filters.
private static final EnumSet<NirCamFilter> SWB_FILTERS = EnumSet.of(
        NirCamFilter.F140M,
        NirCamFilter.F182M,
        NirCamFilter.F210M,
        NirCamFilter.F187N,
        NirCamFilter.F212N);
private static final EnumSet<NirCamFilter> F210R_FILTERS = EnumSet.of(
        NirCamFilter.F210M,
        NirCamFilter.F212N);
private static final EnumSet<NirCamFilter> LWB_FILTERS = EnumSet.of(
        NirCamFilter.F250M,
        NirCamFilter.F300M,
        NirCamFilter.F335M,
        NirCamFilter.F360M,
        NirCamFilter.F410M,
        NirCamFilter.F430M,
        NirCamFilter.F460M,
        NirCamFilter.F480M);
private static final EnumSet<NirCamFilter> F335R_FILTERS = EnumSet.of(
        NirCamFilter.F300M,
        NirCamFilter.F335M);
```

```
private static final EnumSet<NirCamFilter> F430R_FILTERS = EnumSet.of(
        NirCamFilter.F410M,
        NirCamFilter.F430M);

public enum NirCamMask {
    MASK210R("Round", NirCamModule.A, NirCamWavelength.SHORT, NirCamFilter.F182M, NirCamSubarray.SUB128, F210R_FILTERS),
    MASKSWB("Bar", NirCamModule.B, NirCamWavelength.SHORT, NirCamFilter.F182M, NirCamSubarray.SUB128, SWB_FILTERS),
    MASK335R("Round", NirCamModule.A, NirCamWavelength.LONG, NirCamFilter.F335M, NirCamSubarray.SUB64, F335R_FILTERS),
    MASK430R("Round", NirCamModule.A, NirCamWavelength.LONG, NirCamFilter.F335M, NirCamSubarray.SUB64, F430R_FILTERS),
    MASKLWB("Bar", NirCamModule.B, NirCamWavelength.LONG, NirCamFilter.F335M, NirCamSubarray.SUB64, LWB_FILTERS);

    //--------------------Fields--------------------
    String shape;
    NirCamModule module;
    NirCamWavelength camera;
    NirCamFilter tacqFilter;
    NirCamSubarray subarray;
    EnumSet<NirCamFilter> legalFilters;

    //--------------------Constructors--------------------
    NirCamMask(String iShape, NirCamModule iModule, NirCamWavelength iCamera,
            NirCamFilter iTacqFilter, NirCamSubarray iSubarray, EnumSet<NirCamFilter> iLegalFilters) {
        shape = iShape;
        module = iModule;
        camera = iCamera;
        tacqFilter = iTacqFilter;
        subarray = iSubarray;
        legalFilters = iLegalFilters;
    }

    //--------------------Accessors--------------------
    public String getShape() {
        return shape;
    }

    public NirCamModule getModule() {
        return module;
    }

    public NirCamWavelength getCamera() {
        return camera;
    }

    public NirCamFilter getTacqFilter() {
        return tacqFilter;
    }

    public NirCamSubarray getSubarray() {
        return subarray;
    }
```

```
        public EnumSet<NirCamFilter> getLegalFilters () {
            return legalFilters;
        }
    }

    //Mechanism Type
    public enum NirCamMechanism {
        FILTER,
        PUPIL
    }

    //Wheel
    public enum NirCamWheel {
        SHORTA,
        SHORTB,
        LONGA,
        LONGB,
        ALL
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template;

import java.util.List;

/**
 * JwstTemplateFactory - a registry of all the JWST Templates which will control which ones to
 * allow with each instrument.
 *
 * @author Rob Douglas
 */
public enum JwstTemplateFactory {
    //--------------------Statics--------------------
    //MIRI
    MIRI_IMAGING("MIRI Imaging", MiriInstrument.getInstance()) {
        @Override
        public MiriImagingTemplate makeInstance() {
            return new MiriImagingTemplate();
        }
    },
    MIRI_LRS("MIRI Low Resolution Spectroscopy", MiriInstrument.getInstance()) {
        @Override
        public MiriLrsTemplate makeInstance() {
            return new MiriLrsTemplate();
        }
    },
    MIRI_MRS("MIRI Medium Resolution Spectroscopy", MiriInstrument.getInstance()){
        @Override
        public MiriMrsTemplate makeInstance() {
            return new MiriMrsTemplate();
        }
    },
    MIRI_CORON("MIRI Coronagraphic Imaging", MiriInstrument.getInstance()){
        @Override
        public MiriCoronTemplate makeInstance() {
            return new MiriCoronTemplate();
        }
    },
    MIRI_DARK("MIRI Dark", MiriInstrument.getInstance()){
        @Override
        public MiriDarkTemplate makeInstance() {
            return new MiriDarkTemplate();
        }
    },
    MIRI_IMG_FLAT("MIRI Imaging Flat", MiriInstrument.getInstance()){
        @Override
        public MiriImagingFlatTemplate makeInstance() {
            return new MiriImagingFlatTemplate();
        }
    },
```

```
    MIRI_MRS_FLAT("MIRI MRS Flat", MiriInstrument.getInstance()){
        @Override
        public MiriMrsFlatTemplate makeInstance() {
            return new MiriMrsFlatTemplate();
        }
    },
    MIRI_ANNEAL("MIRI Anneal", MiriInstrument.getInstance()){
        @Override
        public MiriAnnealTemplate makeInstance() {
            return new MiriAnnealTemplate();
        }
    },
    //NIRCAM
    NIRCAM_IMAGING("NIRCam Imaging", NirCamInstrument.getInstance()) {
        @Override
        public NirCamImagingTemplate makeInstance() {
            return new NirCamImagingTemplate();
        }
    },
    NIRCAM_CORON("NIRCam Coronagraphic Imaging", NirCamInstrument.getInstance()){
        @Override
        public NirCamCoronTemplate makeInstance() {
            return new NirCamCoronTemplate();
        }
    },
    NIRCAM_DARK("NIRCam Dark", NirCamInstrument.getInstance()){
        @Override
        public NirCamDarkTemplate makeInstance() {
            return new NirCamDarkTemplate();
        }
    },
    NIRCAM_FLAT("NIRCam Internal Flat", NirCamInstrument.getInstance()){
        @Override
        public NirCamFlatTemplate makeInstance() {
            return new NirCamFlatTemplate();
        }
    },
    NIRCAM_WHEEL_EXERCISE("NIRCam Wheel Exercise", NirCamInstrument.getInstance()){
        @Override
        public NirCamWheelExerciseTemplate makeInstance() {
            return new NirCamWheelExerciseTemplate();
        }
    },
    NIRCAM_FOCUS("NIRCam Focus", NirCamInstrument.getInstance()){
        @Override
        public NirCamFocusTemplate makeInstance() {
            return new NirCamFocusTemplate();
        }
    },
    NIRCAM_WHEEL_THRESHOLD_CURRENT("NIRCam Wheel Threshold Current", NirCamInstrument.getInstance()){
```

```
        @Override
        public NirCamWheelThresholdCurrentTemplate makeInstance() {
            return new NirCamWheelThresholdCurrentTemplate();
        }
    }
    //NIRSPEC
    //TFI
    ;

    //--------------------Fields--------------------
    private final String templateName;
    private final JwstInstrument instrument;

    //--------------------Constructors--------------------
    JwstTemplateFactory(String iTemplateName, JwstInstrument iInstrument){
        this.templateName = iTemplateName;
        this.instrument = iInstrument;
    }

    //--------------------Accessors--------------------
    public String getTemplateName() {return templateName;}

    public JwstInstrument getInstrument() {return instrument;}

    //--------------------Methods--------------------
    public abstract JwstTemplate<? extends JwstInstrument> makeInstance();

    public static List<JwstTemplateFactory> factoriesForIntrument(JwstInstrument iInstrument) {
        List<JwstTemplateFactory> lLegalFactories = new Vector<JwstTemplateFactory>();
        if (iInstrument != null) {
            for(JwstTemplateFactory lFactory : values()) {
                if (iInstrument == lFactory.instrument) {
                    lLegalFactories.add(lFactory);
                }
            }

        }
        return lLegalFactories;
    }

    @Override
    public String toString() {
        return getTemplateName();
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.Arrays;

/**
 * NirCamTemplateFieldFactory - Provides a standard way to access the properties of a NIRCam
 * template that are shared among them.  Its only purpose is to return Fields that can be
 * used within the Templates.
 *
 * @author Rob Douglas
 */
public class NirCamTemplateFieldFactory {
    //--------------------Statics--------------------
    public static final String MODULE = "Module";
    public static final String FLAT_SUITE = "Flat Suite";
    public static final String WAVELENGTH = "Wavelength";
    public static final String REQUESTED_EXPOSURE_TIME = "Requested ExpTime";
    public static final String NUMBER_OF_EXPS = "No. of Exposures";
    public static final String NUMBER_OF_GROUPS = "No. of Groups";
    public static final String NUMBER_OF_INTS = "No. of Integrations";
    public static final String ACTUAL_EXP_TIME = "Actual Exp Time";
    public static final String SUBARRAY = "Subarray";
    public static final String READOUT_PATTERN = "Readout Pattern";
    public static final String OBJECT_TYPE = "Object Type";
    public static final String FILTERS = "Filters";
    public static final String EXPOSURES = "Exposures";
    public static final String LINEAR_ACTUATORS = "Linear Actuator Positions";
    public static final String FILTER = "Filter";
    public static final String SHORT_FILTER = "Short Filter";
    public static final String LONG_FILTER = "Long Filter";
    public static final String PUPIL = "Pupil";
    public static final String SHORT_PUPIL = "Short Pupil";
    public static final String LONG_PUPIL = "Long Pupil";
    public static final String MASK = "Coronagraphic Mask";
    public static final String CORON_FILTER = "Coron Mask/Filter";
    public static final String MECH_TYPE = "Mechanism Type";
    public static final String WHEEL = "Wheel";
    public static final String NUMBER_OF_ROTATIONS = "No. of Rotations";
    public static final String RETURN_TO_START = "Return To Start";

    private static final int LINEAR_ACTUATOR_MIN_STEPS = -11900;
    private static final int LINEAR_ACTUATOR_MAX_STEPS = 11900;
    private static final int LINEAR_ACTUATOR_MIN_SENSOR_UNITS = -32767;
    private static final int LINEAR_ACTUATOR_MAX_SENSOR_UNITS = 32767;

    //--------------------Constructors--------------------
    //Singleton - this is a static only class
    private NirCamTemplateFieldFactory() {}
```

```
//--------------------Methods--------------------
/*
 *
 */
public static CosiConstrainedSelection<NirCamModule> makeModuleField (JwstTemplate<NirCamInstrument> iIU) {
    Collection<NirCamModule> lDetectors = Arrays.asList(NirCamModule.values());
    return new CosiConstrainedSelection<NirCamModule>(iIU,MODULE, lDetectors, true);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamSubarray> makeSubarrayField (JwstTemplate<NirCamInstrument> iIU) {
    Collection<NirCamSubarray> lSubarrays = Arrays.asList(NirCamSubarray.values());
    return new CosiConstrainedSelection<NirCamSubarray>(iIU,SUBARRAY, lSubarrays, true);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamReadPattern> makeReadoutPatternField (TinaDocumentElement iContainer) {
    Collection<NirCamReadPattern> lReadPatterns = Arrays.asList(NirCamReadPattern.values());
    return new CosiConstrainedSelection<NirCamReadPattern>(iContainer,READOUT_PATTERN, lReadPatterns, true);
}

/*
 *
 */
public static NirCamImagingExposureSpecTable makeNirCamImagingExposureTableField (JwstTemplate<NirCamInstrument> iIU) {
    return new NirCamImagingExposureSpecTable(iIU, FILTERS);
}

/*
 *
 */
public static NirCamCoronExposureSpecTable makeNirCamCoronExposureTableField (JwstTemplate<NirCamInstrument> iIU) {
    return new NirCamCoronExposureSpecTable(iIU, FILTERS);
}

/*
 *
 */
public static NirCamDarkExposureSpecTable makeNirCamDarkExposureTableField (JwstTemplate<NirCamInstrument> iIU) {
    return new NirCamDarkExposureSpecTable(iIU, EXPOSURES);
}

/*
 *
 */
public static NirCamFlatExposureSpecTable makeNirCamFlatExposureTableField (JwstTemplate<NirCamInstrument> iIU) {
```

```
        return new NirCamFlatExposureSpecTable(iIU, FILTERS);
    }

    /*
     *
     */
    public static NirCamLinearActuatorTable makeNirCamLinearActuatorTableField (JwstTemplate<NirCamInstrument> iIU) {
        return new NirCamLinearActuatorTable(iIU, LINEAR_ACTUATORS);
    }

    /*
     *
     */
    public static CosiConstrainedDouble makeRequestedExpTimeField (TinaDocumentElement iContainer) {
        return new CosiConstrainedDouble(iContainer, REQUESTED_EXPOSURE_TIME, true, 0.0, Double.MAX_VALUE);
    }

    /*
     *
     */
    public static CosiConstrainedInt makeNumberOfExpsField (TinaDocumentElement iContainer) {
        CosiConstrainedInt lNumExps = new CosiConstrainedInt(iContainer, NUMBER_OF_EXPS, true, 1, Integer.MAX_VALUE);
        lNumExps.set(1);
        return lNumExps;
    }

    /*
     *
     */
    public static CosiConstrainedInt makeNumberOfGroupsField (TinaDocumentElement iContainer) {
        CosiConstrainedInt lNumGroups = new CosiConstrainedInt(iContainer, NUMBER_OF_GROUPS, true, 1, Integer.MAX_VALUE);
        lNumGroups.set(0);
        return lNumGroups;
    }

    /*
     *
     */
    public static CosiConstrainedInt makeNumberOfIntsField (TinaDocumentElement iContainer) {
        CosiConstrainedInt lNumInts = new CosiConstrainedInt(iContainer, NUMBER_OF_INTS, true, 1, Integer.MAX_VALUE);
        lNumInts.set(0);
        return lNumInts;
    }

    /*
     *
     */
    public static CosiConstrainedSelection<NirCamFilter> makeShortFilterField (TinaDocumentElement iContainer) {
        return makeShortFilterField(iContainer, SHORT_FILTER);
    }
```

```
/*
 *
 */
public static CosiConstrainedSelection<NirCamFilter> makeShortFilterField (TinaDocumentElement iContainer, String iName) {
    List<NirCamFilter> lFilters = NirCamFilter.getFilters(NirCamWavelength.SHORT);
    return makeFilterField(iContainer, iName, lFilters);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamFilter> makeLongFilterField (TinaDocumentElement iContainer) {
    return makeLongFilterField(iContainer, LONG_FILTER);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamFilter> makeLongFilterField (TinaDocumentElement iContainer, String iName) {
    List<NirCamFilter> lFilters = NirCamFilter.getFilters(NirCamWavelength.LONG);
    return makeFilterField(iContainer, iName, lFilters);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamFilter> makeFilterField (TinaDocumentElement iContainer, String iName,
        List<NirCamFilter> iLegalFilters) {
    return new CosiConstrainedSelection<NirCamFilter>(iContainer, iName, iLegalFilters, true);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamFilter> makeAcqFilterField (JwstTemplate<NirCamInstrument> iIU, String iName) {
    Collection<NirCamFilter> lFilters = Arrays.asList(NirCamInstrument.ACQ_FILTERS);
    return new CosiConstrainedSelection<NirCamFilter>(iIU, iName, lFilters, true);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamPupil> makeShortPupilField (TinaDocumentElement iContainer) {
    return makeShortPupilField(iContainer, SHORT_PUPIL);
}

/*
 *
 */
```

```
public static CosiConstrainedSelection<NirCamPupil> makeShortPupilField (TinaDocumentElement iContainer, String iName) {
    List<NirCamPupil> lPupils = NirCamPupil.getPupils(NirCamWavelength.SHORT);
    return makePupilField(iContainer, iName, lPupils);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamPupil> makeLongPupilField (TinaDocumentElement iContainer) {
    return makeLongPupilField(iContainer, LONG_PUPIL);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamPupil> makeLongPupilField (TinaDocumentElement iContainer, String iName) {
    List<NirCamPupil> lPupils = NirCamPupil.getPupils(NirCamWavelength.LONG);
    return makePupilField(iContainer, iName, lPupils);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamPupil> makePupilField (TinaDocumentElement iContainer, String iName,
        List<NirCamPupil> iLegalPupils) {
    return new CosiConstrainedSelection<NirCamPupil>(iContainer, iName, iLegalPupils, true);
}

/*
 *
 */
public static CosiConstrainedSelection<NirCamMask> makeMaskField (JwstTemplate<NirCamInstrument> iIU) {
    Collection<NirCamMask> lMasks = Arrays.asList(NirCamMask.values());
    return new CosiConstrainedSelection<NirCamMask>(iIU, MASK, lMasks, true);
}

/*
 *
 */
public static CosiConstrainedDouble makeActualExpTimeField (TinaDocumentElement iContainer) {
    CosiConstrainedDouble lField =
        new CosiConstrainedDouble(iContainer, ACTUAL_EXP_TIME, false, 0.0, Double.MAX_VALUE);
    lField.setEditable(false);
    return lField;
}

public static CosiConstrainedSelection<NirCamMechanism> makeMechTypeField(JwstTemplate<NirCamInstrument> iIU) {
    Collection<NirCamMechanism> lMech = Arrays.asList(NirCamMechanism.values());
    return new CosiConstrainedSelection<NirCamMechanism>(iIU, MECH_TYPE, lMech, true);
}
```

```
    public static CosiConstrainedMultiSelection<NirCamWheel> makeWheelField(JwstTemplate<NirCamInstrument> iIU) {
        Collection<NirCamWheel> lMechs = Arrays.asList(NirCamWheel.values());
        CosiConstrainedMultiSelection<NirCamWheel> lField =
            new CosiConstrainedMultiSelection<NirCamWheel>(iIU, WHEEL, true, 1, 4, UIdGenerator.DEFAULT_UIDGENERATOR);
        lField.setLegalValues(lMechs);
        return lField;
    }

    public static CosiConstrainedInt makeNumberOfRotationsField(JwstTemplate<NirCamInstrument> iIU) {
        CosiConstrainedInt lField =
            new CosiConstrainedInt(iIU, NUMBER_OF_ROTATIONS, true, 1, 6);
        return lField;
    }

    /*
     * Multiple of these fields are needed, three for the starting position and three for each
     * intermediate position chosen.  The names are therefore passed in separately.
     */
    public static CosiConstrainedInt makeLinearActuatorStepsField(TinaDocumentElement iContainer, String iName) {
        return new CosiConstrainedInt(iContainer, iName, true, LINEAR_ACTUATOR_MIN_STEPS, LINEAR_ACTUATOR_MAX_STEPS);
    }

    public static CosiConstrainedInt makeLinearActuatorSensorUnitsField(NirCamFocusTemplate iIU, String iName) {
        return new CosiConstrainedInt(iIU, iName, true, LINEAR_ACTUATOR_MIN_SENSOR_UNITS, LINEAR_ACTUATOR_MAX_SENSOR_UNITS);
    }

    public static CosiConstrainedInt makeLinearActuatorMotorPhaseField(NirCamFocusTemplate iIU, String iName) {
        return new CosiConstrainedInt(iIU, iName, true, 1, 6);
    }

    public static CosiBooleanField makeFocusReturnToStartField (NirCamFocusTemplate iIU) {
        return new CosiBooleanField(iIU, RETURN_TO_START, true);
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import org.jdom.Element;

/**
 * NirCamExposureSpecification - configures the shared properties of all ways of expressing
 * exposure data in a NirCam template, including access to the Derived parameter calculations.
 *
 * @author Rob Douglas
 */
public abstract class NirCamExposureSpecification extends AbstractTinaDocumentElement {
    //-------------------Statics-------------------
    private static final String EXPOSURE_TIME = "Exposure Time";

    //--------------------Fields--------------------
    protected final CosiConstrainedSelection<NirCamFilter> shortFilter = NirCamTemplateFieldFactory.makeShortFilterField(this);
    protected final CosiConstrainedSelection<NirCamFilter> longFilter = NirCamTemplateFieldFactory.makeLongFilterField(this);
    protected final CosiConstrainedDouble requestedExpTime = NirCamTemplateFieldFactory.makeRequestedExpTimeField(this); {
        //TODO ROB - this is only temporarily made unusable.  It will hopefully be back one day.
        requestedExpTime.setRequired(false);
        requestedExpTime.setEditable(false);
    }

    protected final CosiConstrainedSelection<NirCamReadPattern> readoutPatternField =
NirCamTemplateFieldFactory.makeReadoutPatternField(this);
    protected final CosiConstrainedInt numberOfGroupsField = NirCamTemplateFieldFactory.makeNumberOfGroupsField(this);
    protected final CosiConstrainedInt numberOfIntegrationsField = NirCamTemplateFieldFactory.makeNumberOfIntsField(this);
    protected final CosiConstrainedDouble actualExpTimeField = NirCamTemplateFieldFactory.makeActualExpTimeField(this);

    //Derived values - work on this later
//    protected final CosiDerivedProperty<MiriReadPattern> readoutPattern =
//        CosiDerivedProperty.createUninitializedProperty(this, null, new CosiReadoutPatternCalculator());
//    protected final CosiDerivedProperty<Integer> numberOfGroups =
//        CosiDerivedProperty.createUninitializedProperty(this, 0, new CosiNumberOfGroupsCalculator());
//    protected final CosiDerivedProperty<Integer> numberOfIntegrations =
//        CosiDerivedProperty.createUninitializedProperty(this, 0, new CosiNumberOfIntegrationsCalculator());
//    protected final CosiDerivedProperty<Double> actualExpTime =
//        CosiDerivedProperty.createUninitializedProperty(this, 0.0, new CosiActualExpTimeCalculator());

    //Tina Group
    protected final TinaFieldGroup expTimeGroup = new TinaFieldGroup(this, EXPOSURE_TIME);
    {
        requestedExpTime.setTinaFieldGroup(expTimeGroup);
        expTimeGroup.setLabel(requestedExpTime, NirCamTemplateFieldFactory.REQUESTED_EXPOSURE_TIME);
        readoutPatternField.setTinaFieldGroup(expTimeGroup);
        expTimeGroup.setLabel(readoutPatternField,NirCamTemplateFieldFactory.READOUT_PATTERN);
        numberOfGroupsField.setTinaFieldGroup(expTimeGroup);
        expTimeGroup.setLabel(numberOfGroupsField,NirCamTemplateFieldFactory.NUMBER_OF_GROUPS);
        numberOfIntegrationsField.setTinaFieldGroup(expTimeGroup);
```

```
    expTimeGroup.setLabel(numberOfIntegrationsField,NirCamTemplateFieldFactory.NUMBER_OF_INTS);
    actualExpTimeField.setTinaFieldGroup(expTimeGroup);
    expTimeGroup.setLabel(actualExpTimeField,NirCamTemplateFieldFactory.ACTUAL_EXP_TIME);
    setProperties(new TinaField[] {shortFilter, longFilter, requestedExpTime, readoutPatternField,
            numberOfGroupsField, numberOfIntegrationsField,
            actualExpTimeField});
}

//--------------------Constructors--------------------
public NirCamExposureSpecification() {
    super();
    configureEditableFields();
    Cosi.completeInitialization(this, NirCamExposureSpecification.class);
}

//--------------------Accessors--------------------
public abstract JwstTemplate<NirCamInstrument> getTemplate();

public abstract NirCamSubarray getSubarray();

public NirCamFilter getShortFilter() {
    return shortFilter.get();
}

public void setShortFilter(NirCamFilter iFilter) {
    shortFilter.set(iFilter);
}

public NirCamFilter getLongFilter() {
    return longFilter.get();
}

public void setLongFilter(NirCamFilter iFilter) {
    longFilter.set(iFilter);
}

public Integer getNumberOfGroups() {
    return numberOfGroupsField.get();
}

public void setNumberOfGroups(Integer iNumGroups) {
    numberOfGroupsField.set(iNumGroups);
}

public void setNumberOfGroups(String iNumGroups) {
    numberOfGroupsField.setValueFromString(iNumGroups);
}

public Integer getNumberOfIntegrations() {
    return numberOfIntegrationsField.get();
```

```
}

public void setNumberOfIntegrations(Integer iIntegrations) {
    numberOfIntegrationsField.set(iIntegrations);
}

public void setNumberOfIntegrations(String iIntegrations) {
    numberOfIntegrationsField.setValueFromString(iIntegrations);
}

public NirCamReadPattern getReadoutPattern() {
    return readoutPatternField.get();
}

public void setReadoutPattern(NirCamReadPattern iReadoutPattern) {
    readoutPatternField.set(iReadoutPattern);
}

public Double getRequestedExpTime() {
    return requestedExpTime.get();
}

public void setRequestedExpTime(Double iRequestedExpTime) {
    requestedExpTime.set(iRequestedExpTime);
}

public void setRequestedExpTime(String iRequestedExpTime) {
    requestedExpTime.setValueFromString(iRequestedExpTime);
}

public Double getActualExpTime() {
    return actualExpTimeField.get();
}

public boolean isCommandingMode() {
    return false;
}

@Override
public String getTypeName() {
    // TODO Auto-generated method stub
    return null;
}

public Element getDomElement() {
    // TODO Auto-generated method stub
    return null;
}

@Override
```

```
    public String toString() {
        return getClass()+": Short Filter="+shortFilter+", Long Filter="+longFilter+", ReqExpTime="+requestedExpTime+",
Read="+readoutPatternField+
        ", NGroups="+numberOfGroupsField+", NInts="+numberOfIntegrationsField;
    }

    //--------------------Methods--------------------
    /*
     * Determines whether certain fields can be edited or should be derived.  Some users
     * are able to override the automatic calculation of derived parameters.
     */
    protected void configureEditableFields() {
        requestedExpTime.setEditable(false);
        actualExpTimeField.setEditable(false);
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import javax.swing.JTable;

/**
 * MiriExposureSpecTable - provides a table interface for defining a list of exposure
 * specification parameters that will be executed for the associated observation.
 *
 * @author Rob Douglas
 */
public abstract class NirCamExposureSpecTable<T extends NirCamExposureSpecTableRow> extends AbstractTinaMultiField<T> {
    //--------------------Fields--------------------
    {
        fDisplayEditButton = false;
    }

    //--------------------Constructors--------------------
    public NirCamExposureSpecTable (TinaDocumentElement iContainer, String iName) {
        super(iContainer, iName);
        Cosi.completeInitialization(this, NirCamExposureSpecTable.class);
    }

    //--------------------Accessors--------------------
    public abstract int getColumnCount ();

    public abstract Class<?> getColumnClass (int iColumn);

    public abstract void setValueAt (Object iValue, int iRow, int iColumn);

    public abstract String getColumnName(int iColumn);

    public abstract Object getValueAt(int rowIndex, int iColumn);

    public JwstTemplate<NirCamInstrument> getTemplate () {
        return (JwstTemplate<NirCamInstrument>)getContainer();
    }

    //--------------------Methods--------------------
    public void configureDefaultEditors (JTable iTable) {
        iTable.setDefaultEditor(CosiConstrainedInt.class, new DefaultTinaCosiFieldEditor());
        iTable.setDefaultEditor(CosiConstrainedDouble.class, new DefaultTinaCosiFieldEditor());
        iTable.setDefaultEditor(CosiConstrainedSelection.class, new CosiConstrainedSelectionEditor());
    }

    public boolean areRowsValid() {
        for(T lRow : getValue()) {
            if(lRow.isRowValid() == false) {
                return false;
            }
```

```
        }
        return true;
    }

    @Override
    public int addField(T iField) {
        int lRetVal = super.addField(iField);
        getTemplate().add(iField, false);
        return lRetVal;
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import edu.stsci.CoSI.Cosi;

/**
 * MiriExposureSpecTableRow - represents a single exposure specification for a MIRI
 * exposure.
 *
 * @author Rob Douglas
 */
public abstract class NirCamExposureSpecTableRow<T extends NirCamExposureSpecTable>
extends NirCamExposureSpecification
implements TinaMultiFieldField {
    //--------------------Constructors--------------------
    public NirCamExposureSpecTableRow () {
        super();
        Cosi.completeInitialization(this,NirCamExposureSpecTableRow.class);
    }

    //--------------------Accessors--------------------
    //TODO ROB - this should be made more generic.
    public boolean isRowValid() {
        return (!shortFilter.isRequired() || shortFilter.isSpecified()) &&
        (!longFilter.isRequired() || longFilter.isSpecified()) &&
        (!requestedExpTime.isRequired() || requestedExpTime.getValue() != null) &&
        readoutPatternField.isSpecified() && !readoutPatternField.isIllegalSelection() &&
        numberOfGroupsField.getValue() != null && !numberOfGroupsField.isOutOfRange() &&
        numberOfIntegrationsField.getValue() != null && !numberOfIntegrationsField.isOutOfRange();
    }
}
```

```java
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.Arrays;


public class NirCamImagingTemplate extends JwstTemplate<NirCamInstrument> {
    private static final NirCamSubarray[] LEGAL_SUBARRAYS = new NirCamSubarray[] {
        NirCamSubarray.FULL,
        NirCamSubarray.SUB16,
        NirCamSubarray.SUB48,
        NirCamSubarray.SUB96,
        NirCamSubarray.SUB320,
        NirCamSubarray.SUB640
    };

    //--------------------Fields--------------------
    private final CosiConstrainedSelection<NirCamModule> module = NirCamTemplateFieldFactory.makeModuleField(this);
    private final CosiConstrainedSelection<NirCamSubarray> subarray = NirCamTemplateFieldFactory.makeSubarrayField(this); {
        subarray.setLegalValues(Arrays.asList(LEGAL_SUBARRAYS));
    }
    private final NirCamImagingExposureSpecTable expTable = NirCamTemplateFieldFactory.makeNirCamImagingExposureTableField(this);
    {
        templateProperties = new TinaField[] {module, subarray, expTable};
    }

    //--------------------Constructors--------------------
    public NirCamImagingTemplate () {
        super(NirCamInstrument.getInstance());
        Cosi.completeInitialization(this, NirCamImagingTemplate.class);
    }

    //--------------------Accessors--------------------
    public NirCamSubarray getSubarray() {
        return subarray.getValue();
    }

    public void setSubarray(NirCamSubarray iSubarray) {
        subarray.setValue(iSubarray);
    }

    public List<NirCamImagingExposureSpecTableRow> getExposures() {
        return expTable.getValue();
    }

    public void addExposure(NirCamImagingExposureSpecTableRow iRow) {
        expTable.addField(iRow);
    }

    public NirCamModule getModule() {
        return module.getValue();
```

```
    }

    public void setModule(NirCamModule iMod) {
        module.setValue(iMod);
    }

}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.security.InvalidParameterException;

/**
 * NirCamImagingExposureSpecTable - provides a table interface for defining a list of exposure
 * specification parameters that will be executed for the associated observation.
 *
 * @author Rob Douglas
 */
public class NirCamImagingExposureSpecTable extends NirCamExposureSpecTable<NirCamImagingExposureSpecTableRow> {
    //--------------------Constructors--------------------
    public NirCamImagingExposureSpecTable (TinaDocumentElement iContainer, String iName) {
        super(iContainer, iName);
        Cosi.completeInitialization(this, NirCamImagingExposureSpecTable.class);
    }

    //--------------------Accessors--------------------
    @Override
    public int getColumnCount () {
        return 7;
    }

    @Override
    public Class<?> getColumnClass (int iColumn) {
        switch (iColumn) {
            case 0: return CosiConstrainedSelection.class;
            case 1: return CosiConstrainedSelection.class;
            case 2: return CosiConstrainedDouble.class;
            case 3: return CosiConstrainedSelection.class;
            case 4: return CosiConstrainedInt.class;
            case 5: return CosiConstrainedInt.class;
            case 6: return CosiConstrainedDouble.class;
            default:
                throw new InvalidParameterException("Expected column less than " + getColumnCount());
        }
    }

    // BE ADVISED!!! There is some inconsistent behavior in the order properties are fired
    // between ConstrainedInt and ConstrainedSelection.  This means that when this method is
    // called, ConstrainedSelection still has its old, original value, but ConstrainedInt
    // has been updated with its new value!
    @Override
    public void setValueAt (Object iValue, int iRow, int iColumn) {
        NirCamImagingExposureSpecTableRow lField = getValue().get(iRow);

        if ((iValue != null) && (!"".equals(iValue.toString()))) {
            switch (iColumn) {
                case 0: lField.setShortFilter((NirCamFilter)iValue); break;
```

```
                case 1: lField.setLongFilter((NirCamFilter)iValue); break;
                case 2: lField.setRequestedExpTime(iValue.toString()); break;
                case 3: lField.setReadoutPattern((NirCamReadPattern)iValue); break;
                case 4: lField.setNumberOfGroups(iValue.toString()); break;
                case 5: lField.setNumberOfIntegrations(iValue.toString()); break;
                case 6: break;
                default:
                    throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
            }
        }
    }

    @Override
    public String getColumnName(int iColumn) {
        switch (iColumn) {
            case 0: return NirCamTemplateFieldFactory.SHORT_FILTER;
            case 1: return NirCamTemplateFieldFactory.LONG_FILTER;
            case 2: return NirCamTemplateFieldFactory.REQUESTED_EXPOSURE_TIME;
            case 3: return NirCamTemplateFieldFactory.READOUT_PATTERN;
            case 4: return NirCamTemplateFieldFactory.NUMBER_OF_GROUPS;
            case 5: return NirCamTemplateFieldFactory.NUMBER_OF_INTS;
            case 6: return NirCamTemplateFieldFactory.ACTUAL_EXP_TIME;
            default:
                throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
        }
    }

    @Override
    public Object getValueAt(int rowIndex, int iColumn) {
        NirCamImagingExposureSpecTableRow lField = getValue().get(rowIndex);
        switch (iColumn) {
            case 0: return lField.shortFilter;
            case 1: return lField.longFilter;
            case 2: return lField.requestedExpTime;
            case 3: return lField.readoutPatternField;
            case 4: return lField.numberOfGroupsField;
            case 5: return lField.numberOfIntegrationsField;
            case 6: return lField.actualExpTimeField;
        }
        throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
    }

    //--------------------Methods--------------------
    @Override
    protected NirCamImagingExposureSpecTableRow newField() {
        NirCamImagingExposureSpecTableRow lRow = new NirCamImagingExposureSpecTableRow();
        getTemplate().add(lRow, false);
        return lRow;
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.Arrays;

/**
 * NirCamImagingExposureSpecTableRow - represents a single exposure specification for a NirCam
 * Imaging exposure.
 *
 * @author Rob Douglas
 */
public class NirCamImagingExposureSpecTableRow extends NirCamExposureSpecTableRow<NirCamImagingExposureSpecTable>
implements TinaMultiFieldField {
    //--------------------Statics--------------------
    private static final NirCamFilter[] LEGAL_SHORTS = new NirCamFilter[] {
        NirCamFilter.F070W,
        NirCamFilter.F090W,
        NirCamFilter.F115W,
        NirCamFilter.F150W,
        NirCamFilter.F150W2,
        NirCamFilter.F200W,
        NirCamFilter.F140M,
        NirCamFilter.F182M,
        NirCamFilter.F210M,
        NirCamFilter.F187N,
        NirCamFilter.F212N,
        NirCamFilter.F162M_F150W2,
        NirCamFilter.F164N_F150W,
        NirCamFilter.F164N_F150W2,
        NirCamFilter.F225N_F150W2
    };
    private static final NirCamFilter[] LEGAL_LONGS = new NirCamFilter[] {
        NirCamFilter.F277W,
        NirCamFilter.F322W2,
        NirCamFilter.F356W,
        NirCamFilter.F444W,
        NirCamFilter.F250M,
        NirCamFilter.F300M,
        NirCamFilter.F335M,
        NirCamFilter.F360M,
        NirCamFilter.F410M,
        NirCamFilter.F430M,
        NirCamFilter.F460M,
        NirCamFilter.F480M,
        NirCamFilter.F323N_F356W,
        NirCamFilter.F323N_F322W2,
        NirCamFilter.F405N_F444W,
        NirCamFilter.F405N_F410M,
        NirCamFilter.F418N_F444W,
        NirCamFilter.F418N_F410M,
```

```
        NirCamFilter.F466N_F444W,
        NirCamFilter.F466N_F460M,
        NirCamFilter.F470N_F444W
    };

    //--------------------Fields--------------------
    {
        shortFilter.setLegalValues(Arrays.asList(LEGAL_SHORTS));
        longFilter.setLegalValues(Arrays.asList(LEGAL_LONGS));
    }

    //--------------------Constructors--------------------
    public NirCamImagingExposureSpecTableRow () {
        super();
        //filter.setLegalValues(Arrays.asList(LEGAL_IMAGING_FILTERS));
        Cosi.completeInitialization(this,NirCamImagingExposureSpecTableRow.class);
    }

    //--------------------Accessors--------------------
    @Override
    public NirCamImagingTemplate getTemplate() {
        return (NirCamImagingTemplate)getParent();
    }

    @Override
    public NirCamSubarray getSubarray() {
        return (getTemplate()).getSubarray();
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.ArrayList;

/**
 * NirCamTargetAcqTemplate - This class encapsulates the properties shared by templates that
 * need to do a special target acquisition.
 *
 * @author Rob Douglas
 */
public abstract class NirCamTargetAcqTemplate extends JwstTemplate<NirCamInstrument> {
    //--------------------Statics--------------------
    private static final String ACQ_TARGET = "AcqTarget";
    private static final String ACQ_FILTER = "AcqFilter";
    private static final String ACQ_FLUX = "AcqFlux";
    private static final String TACQ_FIELDS = "Target ACQ";

    //--------------------Fields--------------------
    protected final AutoConstrainedSelection<NumberedTarget> acqTargetChooser =
        CosiAutoConstrainedSelectionField.makeInstance(this, ACQ_TARGET, true, new CosiLegalTargetsCalculator(),
NumberedTarget.NUMBERED_TARGET_UIdGenerator);
    protected final CosiConstrainedSelection<NirCamFilter> acqFilter = NirCamTemplateFieldFactory.makeAcqFilterField(this,
ACQ_FILTER);
    protected final CosiConstrainedDouble acqFluxField = new CosiConstrainedDouble(this, ACQ_FLUX, true, 0.0, Double.MAX_VALUE);
    protected final CosiDerivedProperty<Double> acqFlux = CosiDerivedProperty.createUninitializedProperty(this, 0.0, new
CosiAcqFluxCalculator());
    protected final TinaFieldGroup tacqFields = new TinaFieldGroup(this, TACQ_FIELDS);
    {
        acqTargetChooser.setTinaFieldGroup(tacqFields);
        tacqFields.setLabel(acqTargetChooser, ACQ_TARGET);
        acqFilter.setTinaFieldGroup(tacqFields);
        tacqFields.setLabel(acqFilter, ACQ_FILTER);
        acqFluxField.setTinaFieldGroup(tacqFields);
        acqFluxField.setEditable(false);
        tacqFields.setLabel(acqFluxField, ACQ_FLUX);
        templateProperties = new TinaField[] {acqTargetChooser, acqFilter, acqFluxField};
    }

    //--------------------Constructors--------------------
    public NirCamTargetAcqTemplate(NirCamInstrument iInstrument) {
        super(iInstrument);
        Cosi.completeInitialization(this, NirCamTargetAcqTemplate.class);
    }

    //--------------------Accessors--------------------
    public Target getAcqTarget() {
        return acqTargetChooser.get();
    }
```

```
public void setTarget(String iID) {
    acqTargetChooser.setValueFromString(iID);
}

public String getAcqTargetAsString() {
    return acqTargetChooser.getValueAsString();
}

public void setAcqTargetFromString(String iTarg) {
    acqTargetChooser.setValueFromString(iTarg);
}

public NirCamFilter getAcqFilter() {
    return acqFilter.get();
}

public void setAcqFilter (NirCamFilter iFilter) {
    acqFilter.set(iFilter);
}

public Double getAcqFlux() {
    return acqFluxField.get();
}

public void setAcqFlux (Double iFlux) {
    acqFluxField.set(iFlux);
}

//-------------------- Constraints --------------------
@CosiConstraint
private void cosiSetActualExpTime () {
    acqFluxField.set(acqFlux.get());
}

//--------------------Inner Classes--------------------
private final class CosiLegalTargetsCalculator implements Calculator<Collection<NumberedTarget>> {
    public Collection<NumberedTarget> calculate() {
        return getTinaDocument() == null ? new ArrayList<NumberedTarget>() :
            getTinaDocument().getChildren(NumberedTarget.class, ALL);
    }
}

private class CosiAcqFluxCalculator implements Calculator<Double> {
    public Double calculate() {
        NumberedTarget lTarg = acqTargetChooser.get();
        if (lTarg!=null && lTarg instanceof JwstFixedTarget) {
            return ((JwstFixedTarget)lTarg).getFlux(getInstrument(), getAcqFilter());
        }
        return null;
    }
```

```
    }
}
```

```java
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.List;

public class NirCamCoronTemplate extends NirCamTargetAcqTemplate {
    //--------------------Fields--------------------
    private final CosiConstrainedSelection<NirCamMask> mask = NirCamTemplateFieldFactory.makeMaskField(this);
    private CosiDerivedProperty<Boolean> useShortFilter =
        CosiDerivedProperty.createUninitializedProperty(this, Boolean.TRUE, new CosiShortFilterCalculator());
    private final NirCamCoronExposureSpecTable expTable = NirCamTemplateFieldFactory.makeNirCamCoronExposureTableField(this);
    {
        templateProperties = (TinaField<?>[])ArrayUtils.addArrays(new TinaField[] {mask}, templateProperties);
        templateProperties = (TinaField<?>[])ArrayUtils.addArrays(templateProperties, new TinaField[] {expTable});
    }
    //--------------------Constructors--------------------
    public NirCamCoronTemplate () {
        super(NirCamInstrument.getInstance());
        acqFilter.setEditable(false);
        Cosi.completeInitialization(this, NirCamCoronTemplate.class);
    }

    //--------------------Accessors--------------------

    public NirCamMask getMask() {
        return mask.get();
    }

    public void setMask(NirCamMask iMask) {
        mask.set(iMask);
    }

    public Boolean useShortFilter() {
        return useShortFilter.get();
    }

    //--------------------Constraints--------------------
    @CosiConstraint
    private void cosiSetAcqFilter () {
        NirCamFilter lFilter = null;
        if (mask.isSpecified()) {
            lFilter = getMask().getTacqFilter();
        }
        setAcqFilter(lFilter);
    }

    public List<NirCamCoronExposureSpecTableRow> getExposures() {
        return expTable.getValue();
    }

    public void addExposure(NirCamCoronExposureSpecTableRow iRow) {
```

```
        expTable.addField(iRow);
    }

    //--------------------Inner Classes--------------------
    private final class CosiShortFilterCalculator implements Calculator<Boolean> {
        public Boolean calculate () {
            NirCamMask lMask = getMask();
            if (lMask!=null) {
                NirCamWavelength lCamera = lMask.getCamera();
                if (lCamera!=null) {
                    return (lCamera == NirCamWavelength.SHORT);
                }
            }
            return true;
        }
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.security.InvalidParameterException;

/**
 * NirCamCoronExposureSpecTable - provides a table interface for defining a list of exposure
 * specification parameters that will be executed for the associated observation.
 *
 * @author Rob Douglas
 */
public class NirCamCoronExposureSpecTable extends NirCamExposureSpecTable<NirCamCoronExposureSpecTableRow> {
    //--------------------Statics--------------------
    private static final int MAX_FILTERS = 6;
    private static final String MAX_FILTER_DIAG = "No more than "+MAX_FILTERS+" filters may be used.";

    //--------------------Constructors--------------------
    public NirCamCoronExposureSpecTable (TinaDocumentElement iContainer, String iName) {
        super(iContainer, iName);
        Cosi.completeInitialization(this, NirCamCoronExposureSpecTable.class);
    }

    //--------------------Accessors--------------------
    @Override
    public NirCamCoronTemplate getTemplate() {
        return (NirCamCoronTemplate)super.getTemplate();
    }

    @Override
    public int getColumnCount () {
        return 5;
    }

    @Override
    public Class<?> getColumnClass (int iColumn) {
        switch (iColumn) {
            case 0: return CosiConstrainedSelection.class;
            case 1: return CosiConstrainedSelection.class;
            case 2: return CosiConstrainedInt.class;
            case 3: return CosiConstrainedInt.class;
            case 4: return CosiConstrainedDouble.class;
            default:
                throw new InvalidParameterException("Expected column less than " + getColumnCount());
        }
    }

    // BE ADVISED!!! There is some inconsistent behavior in the order properties are fired
    // between ConstrainedInt and ConstrainedSelection.  This means that when this method is
    // called, ConstrainedSelection still has its old, original value, but ConstrainedInt
    // has been updated with its new value!
```

```
@Override
public void setValueAt (Object iValue, int iRow, int iColumn) {
    NirCamCoronExposureSpecTableRow lField = getValue().get(iRow);

    if ((iValue != null) && (!"".equals(iValue.toString()))) {
        switch (iColumn) {
            case 0:
                if (getTemplate().useShortFilter()) {
                    lField.setShortFilter((NirCamFilter)iValue);
                    lField.setLongFilter(null);
                } else {
                    lField.setShortFilter(null);
                    lField.setLongFilter((NirCamFilter)iValue);
                }
                break;
            case 1: lField.setReadoutPattern((NirCamReadPattern)iValue); break;
            case 2: lField.setNumberOfGroups(iValue.toString()); break;
            case 3: lField.setNumberOfIntegrations(iValue.toString()); break;
            case 4: break;
            default:
                throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
        }
    }
}

@Override
public String getColumnName(int iColumn) {
    switch (iColumn) {
        case 0: return NirCamTemplateFieldFactory.FILTER;
        case 1: return NirCamTemplateFieldFactory.READOUT_PATTERN;
        case 2: return NirCamTemplateFieldFactory.NUMBER_OF_GROUPS;
        case 3: return NirCamTemplateFieldFactory.NUMBER_OF_INTS;
        case 4: return NirCamTemplateFieldFactory.ACTUAL_EXP_TIME;
        default:
            throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
    }
}

@Override
public Object getValueAt(int rowIndex, int iColumn) {
    NirCamCoronExposureSpecTableRow lField = getValue().get(rowIndex);
    switch (iColumn) {
        case 0:
            if (getTemplate().useShortFilter()) {
                return lField.shortFilter;
            } else {
                return lField.longFilter;
            }
        case 1: return lField.readoutPatternField;
        case 2: return lField.numberOfGroupsField;
```

```
            case 3: return lField.numberOfIntegrationsField;
            case 4: return lField.actualExpTimeField;
        }
        throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
    }

    //--------------------Methods--------------------
    @Override
    protected NirCamCoronExposureSpecTableRow newField() {
        NirCamCoronExposureSpecTableRow lRow = new NirCamCoronExposureSpecTableRow();
        getTemplate().add(lRow, false);
        return lRow;
    }

    //--------------------Constraints--------------------
    @CosiConstraint
    private void cosiCheckMaxFiltersUsed () {
        int lNumRows = getRowCount();
        DiagnosticManager.ensureDiagnostic(this, this, this,
                Diagnostic.ERROR,
                MAX_FILTER_DIAG,
                MAX_FILTER_DIAG,
                lNumRows>MAX_FILTERS);
    }

}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.EnumSet;

/**
 * NirCamCoronExposureSpecTableRow - represents a single exposure specification for a NirCam
 * Coronagraphic exposure.
 *
 * @author Rob Douglas
 */
public class NirCamCoronExposureSpecTableRow extends NirCamExposureSpecTableRow<NirCamCoronExposureSpecTable>
implements TinaMultiFieldField {
    //--------------------Constructors--------------------
    public NirCamCoronExposureSpecTableRow () {
        super();
        //filter.setLegalValues(Arrays.asList(LEGAL_IMAGING_FILTERS));
        //The Readout Pattern and Number of Groups are always the same, so they
        //are set here.
        setReadoutPattern(NirCamReadPattern.RAPID);
        setNumberOfGroups(1);
        Cosi.completeInitialization(this,NirCamCoronExposureSpecTableRow.class);
    }

    //--------------------Accessors--------------------
    @Override
    public NirCamCoronTemplate getTemplate() {
        return (NirCamCoronTemplate)getParent();
    }

    @Override
    public NirCamSubarray getSubarray() {
        return null;
    }

    public NirCamFilter getFilter() {
        if (getTemplate().useShortFilter()) {
            return getShortFilter();
        } else {
            return getLongFilter();
        }
    }

    public void setFilter(NirCamFilter iFilter) {
        if (getTemplate().useShortFilter()) {
            setShortFilter(iFilter);
        } else {
            setLongFilter(iFilter);
        }
    }
```

```
    //--------------------Constraints--------------------
    @CosiConstraint
    private void cosiSetLegalFilters () {
        if (getTemplate() != null) {
            NirCamMask lMask = getTemplate().getMask();
            if (lMask!=null) {
                EnumSet<NirCamFilter> lLegalFilters = lMask.getLegalFilters();
                switch (lMask.getCamera()) {
                    case SHORT:
                        shortFilter.setRequired(true);
                        shortFilter.setLegalValues(lLegalFilters);
                        longFilter.setRequired(false);
                        longFilter.setLegalValues(new Vector<NirCamFilter>());
                        break;
                    case LONG:
                        shortFilter.setRequired(false);
                        shortFilter.setLegalValues(new Vector<NirCamFilter>());
                        longFilter.setLegalValues(lLegalFilters);
                        longFilter.setRequired(true);
                        break;
                    default:
                        break;
                }
            }
        }
    }
}
```

```java
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.Arrays;


public class NirCamDarkTemplate extends JwstTemplate<NirCamInstrument> {
    private static final NirCamSubarray[] LEGAL_SUBARRAYS = new NirCamSubarray[] {
        NirCamSubarray.FULL,
        NirCamSubarray.SUB16,
        NirCamSubarray.SUB48,
        NirCamSubarray.SUB96,
        NirCamSubarray.SUB320,
        NirCamSubarray.SUB640
    };

    //--------------------Fields--------------------
    private final CosiConstrainedSelection<NirCamModule> module = NirCamTemplateFieldFactory.makeModuleField(this);
    private final CosiConstrainedSelection<NirCamSubarray> subarray = NirCamTemplateFieldFactory.makeSubarrayField(this); {
        subarray.setLegalValues(Arrays.asList(LEGAL_SUBARRAYS));
    }
    private final NirCamDarkExposureSpecTable expTable = NirCamTemplateFieldFactory.makeNirCamDarkExposureTableField(this);
    {
        templateProperties = new TinaField[] {module, subarray, expTable};
    }

    //--------------------Constructors--------------------
    public NirCamDarkTemplate () {
        super(NirCamInstrument.getInstance());
        Cosi.completeInitialization(this, NirCamDarkTemplate.class);
    }

    //--------------------Accessors--------------------
    public NirCamSubarray getSubarray() {
        return subarray.getValue();
    }

    public void setSubarray(NirCamSubarray iSubarray) {
        subarray.setValue(iSubarray);
    }

    public List<NirCamDarkExposureSpecTableRow> getExposures() {
        return expTable.getValue();
    }

    public void addExposure(NirCamDarkExposureSpecTableRow iRow) {
        expTable.addField(iRow);
    }

    public NirCamModule getModule() {
        return module.getValue();
```

```
    }

    public void setModule(NirCamModule iMod) {
        module.setValue(iMod);
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.security.InvalidParameterException;

/**
 * NirCamDarkExposureSpecTable - provides a table interface for defining a list of exposure
 * specification parameters that will be executed for the associated observation.
 *
 * @author Rob Douglas
 */
public class NirCamDarkExposureSpecTable extends NirCamExposureSpecTable<NirCamDarkExposureSpecTableRow> {
    //--------------------Constructors--------------------
    public NirCamDarkExposureSpecTable (TinaDocumentElement iContainer, String iName) {
        super(iContainer, iName);
        Cosi.completeInitialization(this, NirCamDarkExposureSpecTable.class);
    }

    //--------------------Accessors--------------------
    @Override
    public int getColumnCount () {
        return 5;
    }

    @Override
    public Class<?> getColumnClass (int iColumn) {
        switch (iColumn) {
            case 0: return CosiConstrainedInt.class;
            case 1: return CosiConstrainedSelection.class;
            case 2: return CosiConstrainedInt.class;
            case 3: return CosiConstrainedInt.class;
            case 4: return CosiConstrainedDouble.class;
            default:
                throw new InvalidParameterException("Expected column less than " + getColumnCount());
        }
    }

    // BE ADVISED!!! There is some inconsistent behavior in the order properties are fired
    // between ConstrainedInt and ConstrainedSelection.  This means that when this method is
    // called, ConstrainedSelection still has its old, original value, but ConstrainedInt
    // has been updated with its new value!
    @Override
    public void setValueAt (Object iValue, int iRow, int iColumn) {
        NirCamDarkExposureSpecTableRow lField = getValue().get(iRow);

        if ((iValue != null) && (!"".equals(iValue.toString()))) {
            switch (iColumn) {
                case 0: lField.setNumberOfExposures(iValue.toString()); break;
                case 1: lField.setReadoutPattern((NirCamReadPattern)iValue); break;
                case 2: lField.setNumberOfGroups(iValue.toString()); break;
```

```
                case 3: lField.setNumberOfIntegrations(iValue.toString()); break;
                case 4: break;
                default:
                    throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
            }
        }
    }

    @Override
    public String getColumnName(int iColumn) {
        switch (iColumn) {
            case 0: return NirCamTemplateFieldFactory.NUMBER_OF_EXPS;
            case 1: return NirCamTemplateFieldFactory.READOUT_PATTERN;
            case 2: return NirCamTemplateFieldFactory.NUMBER_OF_GROUPS;
            case 3: return NirCamTemplateFieldFactory.NUMBER_OF_INTS;
            case 4: return NirCamTemplateFieldFactory.ACTUAL_EXP_TIME;
            default:
                throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
        }
    }

    @Override
    public Object getValueAt(int rowIndex, int iColumn) {
        NirCamDarkExposureSpecTableRow lField = getValue().get(rowIndex);
        switch (iColumn) {
            case 0: return lField.numExps;
            case 1: return lField.readoutPatternField;
            case 2: return lField.numberOfGroupsField;
            case 3: return lField.numberOfIntegrationsField;
            case 4: return lField.actualExpTimeField;
        }
        throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
    }

    //--------------------Methods--------------------
    @Override
    protected NirCamDarkExposureSpecTableRow newField() {
        NirCamDarkExposureSpecTableRow lRow = new NirCamDarkExposureSpecTableRow();
        getTemplate().add(lRow, false);
        return lRow;
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import edu.stsci.CoSI.Cosi;

/**
 * NirCamDarkExposureSpecTableRow - represents a single exposure specification for a NirCam
 * Dark exposure.
 *
 * @author Rob Douglas
 */
public class NirCamDarkExposureSpecTableRow extends NirCamExposureSpecTableRow<NirCamDarkExposureSpecTable>
implements TinaMultiFieldField {
    //--------------------Fields--------------------
    protected final CosiConstrainedInt numExps  = MiriTemplateFieldFactory.makeNumberOfExpsField(this);
    {
        shortFilter.setRequired(false);
        longFilter.setRequired(false);
        requestedExpTime.setRequired(false);
    }

    //--------------------Constructors--------------------
    public NirCamDarkExposureSpecTableRow () {
        super();
        Cosi.completeInitialization(this,NirCamDarkExposureSpecTableRow.class);
    }

    //--------------------Accessors--------------------
    @Override
    public NirCamDarkTemplate getTemplate() {
        return (NirCamDarkTemplate)getParent();
    }

    @Override
    public NirCamSubarray getSubarray() {
        return (getTemplate()).getSubarray();
    }

    public Integer getNumberOfExposures() {
        return numExps.get();
    }

    public void setNumberOfExposures(Integer iNumExps) {
        numExps.set(iNumExps);
    }

    public void setNumberOfExposures(String iNumExps) {
        numExps.setValueFromString(iNumExps);
    }
}
```

```java
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.Arrays;


public class NirCamFlatTemplate extends JwstTemplate<NirCamInstrument> {
    //--------------------Statics--------------------
    private static final NirCamModule[] LEGAL_MODULES = {
        NirCamModule.A,
        NirCamModule.B
    };

    //--------------------Fields--------------------
    private final CosiConstrainedSelection<NirCamModule> module = NirCamTemplateFieldFactory.makeModuleField(this);
    private final NirCamFlatExposureSpecTable expTable = NirCamTemplateFieldFactory.makeNirCamFlatExposureTableField(this);

    {
        module.setLegalValues(Arrays.asList(LEGAL_MODULES));
        templateProperties = new TinaField[] {module, expTable};
    }

    //--------------------Constructors--------------------
    public NirCamFlatTemplate () {
        super(NirCamInstrument.getInstance());
        Cosi.completeInitialization(this, NirCamFlatTemplate.class);
    }

    //--------------------Accessors--------------------
    public NirCamModule getModule() {
        return module.get();
    }

    public void setModule(NirCamModule iModule) {
        module.set(iModule);
    }

    public List<NirCamFlatExposureSpecTableRow> getExposures() {
        return expTable.getValue();
    }

    public void addExposure(NirCamFlatExposureSpecTableRow iRow) {
        expTable.addField(iRow);
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.security.InvalidParameterException;

/**
 * NirCamFlatExposureSpecTable - provides a table interface for defining a list of exposure
 * specification parameters that will be executed for the associated observation.
 *
 * @author Rob Douglas
 */
public class NirCamFlatExposureSpecTable extends NirCamExposureSpecTable<NirCamFlatExposureSpecTableRow> {
    //--------------------Constructors--------------------
    public NirCamFlatExposureSpecTable (TinaDocumentElement iContainer, String iName) {
        super(iContainer, iName);
        Cosi.completeInitialization(this, NirCamFlatExposureSpecTable.class);
    }

    //--------------------Accessors--------------------
    @Override
    public int getColumnCount () {
        return 5;
    }

    @Override
    public Class<?> getColumnClass (int iColumn) {
        switch (iColumn) {
            case 0: return CosiConstrainedSelection.class;
            case 1: return CosiConstrainedSelection.class;
            case 2: return CosiConstrainedInt.class;
            case 3: return CosiConstrainedInt.class;
            case 4: return CosiConstrainedDouble.class;
            default:
                throw new InvalidParameterException("Expected column less than " + getColumnCount());
        }
    }

    // BE ADVISED!!! There is some inconsistent behavior in the order properties are fired
    // between ConstrainedInt and ConstrainedSelection.  This means that when this method is
    // called, ConstrainedSelection still has its old, original value, but ConstrainedInt
    // has been updated with its new value!
    @Override
    public void setValueAt (Object iValue, int iRow, int iColumn) {
        NirCamFlatExposureSpecTableRow lField = getValue().get(iRow);

        if ((iValue != null) && (!"".equals(iValue.toString()))) {
            switch (iColumn) {
                case 0: lField.setShortFilter((NirCamFilter)iValue); break;
                case 1: lField.setLongFilter((NirCamFilter)iValue); break;
                case 2: lField.setNumberOfExposures(iValue.toString()); break;
```

```
                case 3: lField.setNumberOfIntegrations(iValue.toString()); break;
                case 4: break;
                default:
                    throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
            }
        }
    }

    @Override
    public String getColumnName(int iColumn) {
        switch (iColumn) {
            case 0: return NirCamTemplateFieldFactory.SHORT_FILTER;
            case 1: return NirCamTemplateFieldFactory.LONG_FILTER;
            case 2: return NirCamTemplateFieldFactory.NUMBER_OF_EXPS;
            case 3: return NirCamTemplateFieldFactory.NUMBER_OF_INTS;
            case 4: return NirCamTemplateFieldFactory.ACTUAL_EXP_TIME;
            default:
                throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
        }
    }

    @Override
    public Object getValueAt(int rowIndex, int iColumn) {
        NirCamFlatExposureSpecTableRow lField = getValue().get(rowIndex);
        switch (iColumn) {
            case 0: return lField.shortFilter;
            case 1: return lField.longFilter;
            case 2: return lField.numExps;
            case 3: return lField.numberOfIntegrationsField;
            case 4: return lField.actualExpTimeField;
        }
        throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
    }

    //--------------------Methods--------------------
    @Override
    protected NirCamFlatExposureSpecTableRow newField() {
        NirCamFlatExposureSpecTableRow lRow = new NirCamFlatExposureSpecTableRow();
        getTemplate().add(lRow, false);
        return lRow;
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.Arrays;

/**
 * NirCamFlatExposureSpecTableRow - represents a single exposure specification for a NirCam
 * Flat exposure.
 *
 * @author Rob Douglas
 */
public class NirCamFlatExposureSpecTableRow extends NirCamExposureSpecTableRow<NirCamFlatExposureSpecTable>
implements TinaMultiFieldField {
    //--------------------Statics--------------------
    private static final NirCamFilter[] LEGAL_SHORTS = new NirCamFilter[] {
        NirCamFilter.F070W,
        NirCamFilter.F090W,
        NirCamFilter.F115W,
        NirCamFilter.F150W,
        NirCamFilter.F200W,
        NirCamFilter.F210M,
        NirCamFilter.F212N,
        NirCamFilter.F150W2,
        NirCamFilter.WL3,
        NirCamFilter.F140M,
        NirCamFilter.F182M,
        NirCamFilter.F187N
    };
    private static final NirCamFilter[] LEGAL_LONGS = new NirCamFilter[] {
        NirCamFilter.F277W,
        NirCamFilter.F356W,
        NirCamFilter.F444W,
        NirCamFilter.F300M,
        NirCamFilter.F335M,
        NirCamFilter.F360M,
        NirCamFilter.F410M,
        NirCamFilter.F430M,
        NirCamFilter.F460M,
        NirCamFilter.F480M,
        NirCamFilter.F250M,
        NirCamFilter.F322W2
    };

    //--------------------Fields--------------------
    protected final CosiConstrainedInt numExps  = MiriTemplateFieldFactory.makeNumberOfExpsField(this);
    {
        shortFilter.setLegalValues(Arrays.asList(LEGAL_SHORTS));
        longFilter.setLegalValues(Arrays.asList(LEGAL_LONGS));
    }
```

```
    //--------------------Constructors--------------------
    public NirCamFlatExposureSpecTableRow () {
        super();
        //filter.setLegalValues(Arrays.asList(LEGAL_IMAGING_FILTERS));
        //The Readout Pattern and Number of Groups are always the same, so they
        //are set here.
        setReadoutPattern(NirCamReadPattern.RAPID);
        setNumberOfGroups(1);
        requestedExpTime.setRequired(false);
        Cosi.completeInitialization(this,NirCamFlatExposureSpecTableRow.class);
    }

    //--------------------Accessors--------------------
    @Override
    public NirCamFlatTemplate getTemplate() {
        return (NirCamFlatTemplate)getParent();
    }

    @Override
    public NirCamSubarray getSubarray() {
        return null;
    }

    public Integer getNumberOfExposures() {
        return numExps.get();
    }

    public void setNumberOfExposures(Integer iNumExps) {
        numExps.set(iNumExps);
    }

    public void setNumberOfExposures(String iNumExps) {
        numExps.setValueFromString(iNumExps);
    }
}
```

```java
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.Set;

public class NirCamWheelExerciseTemplate extends JwstTemplate<NirCamInstrument> {
    //--------------------Statcis--------------------
    private static final String WHEEL_CHECK = "Allowed Wheels";

    //--------------------Fields--------------------
    private final CosiConstrainedSelection<NirCamMechanism> mechType = NirCamTemplateFieldFactory.makeMechTypeField(this);
    private final CosiConstrainedMultiSelection<NirCamWheel> wheels = NirCamTemplateFieldFactory.makeWheelField(this);
    private final CosiConstrainedInt numRotations = NirCamTemplateFieldFactory.makeNumberOfRotationsField(this);
    {
        templateProperties = new TinaField[] {mechType, wheels, numRotations};
    }

    //--------------------Constructors--------------------
    public NirCamWheelExerciseTemplate () {
        super(NirCamInstrument.getInstance());
        Cosi.completeInitialization(this, NirCamWheelExerciseTemplate.class);
    }

    //--------------------Accessors--------------------
    public NirCamMechanism getMechType() {
        return mechType.get();
    }

    public void setMechType(NirCamMechanism iMech) {
        mechType.set(iMech);
    }

    public Set<NirCamWheel> getWheels() {
        return wheels.get();
    }

    public void setWheels(Set<NirCamWheel> iWheels) {
        wheels.set(iWheels);
    }

    public Integer getNumRotations() {
        return numRotations.get();
    }

    public void setNumRotations(Integer iNumRotations) {
        numRotations.set(iNumRotations);
    }

    //--------------------Constraints--------------------
    @CosiConstraint
    private void cosiSetAllowedWheelSelections () {
```

```
        boolean selectedAll = getWheels().contains(NirCamWheel.ALL);
        if (selectedAll) {
            wheels.setMaxSelections(1);
        } else {
            wheels.setMaxSelections(4);
        }
        DiagnosticManager.ensureDiagnostic(this, WHEEL_CHECK, this,
                Diagnostic.ERROR, "If ALL is selected, no other values are allowed.",
                "Selecting more than one Wheels does them each serially, but selecting ALL does them in parallel.",
                (selectedAll && (getWheels().size()>1)));
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.util.Arrays;

/**
 * NirCamFocusTemplate - allows the movement of the focus wheels.
 *
 * @author Rob Douglas
 */
public class NirCamFocusTemplate extends JwstTemplate<NirCamInstrument> {
    //--------------------Statics--------------------
    private static final String EXPOSURE_TIME = "Exposure Time";
    public static final String START_STEPS = "Starting Position in Steps";
    public static final String START_SENSOR_UNITS = "Starting Position in Sensor Units";
    public static final String START_MOTOR_PHASE = "Starting Motor Phase";
    public static final String LINEAR_ACTUATOR = "Linear Actuator";

    private static final NirCamModule[] LEGAL_MODULES = {
        NirCamModule.A,
        NirCamModule.B
    };
    private static final NirCamFilter[] LEGAL_SHORTS = new NirCamFilter[] {
        NirCamFilter.F070W,
        NirCamFilter.F090W,
        NirCamFilter.F115W,
        NirCamFilter.F150W,
        NirCamFilter.F200W,
        NirCamFilter.F210M,
        NirCamFilter.F212N,
        NirCamFilter.F150W2,
        NirCamFilter.WL3,
        NirCamFilter.F140M,
        NirCamFilter.F182M,
        NirCamFilter.F187N
    };
    private static final NirCamFilter[] LEGAL_LONGS = new NirCamFilter[] {
        NirCamFilter.F277W,
        NirCamFilter.F356W,
        NirCamFilter.F444W,
        NirCamFilter.F300M,
        NirCamFilter.F335M,
        NirCamFilter.F360M,
        NirCamFilter.F410M,
        NirCamFilter.F430M,
        NirCamFilter.F460M,
        NirCamFilter.F480M,
        NirCamFilter.F250M,
        NirCamFilter.F322W2
    };
```

```
    //--------------------Fields--------------------
    private final CosiConstrainedSelection<NirCamModule> module = NirCamTemplateFieldFactory.makeModuleField(this); {
        module.setLegalValues(Arrays.asList(LEGAL_MODULES));
    }
    private final CosiConstrainedSelection<NirCamFilter> shortFilter = NirCamTemplateFieldFactory.makeShortFilterField(this);
    private final CosiConstrainedSelection<NirCamFilter> longFilter = NirCamTemplateFieldFactory.makeLongFilterField(this); {
        shortFilter.setLegalValues(Arrays.asList(LEGAL_SHORTS));
        longFilter.setLegalValues(Arrays.asList(LEGAL_LONGS));
    }
    private final CosiConstrainedSelection<NirCamPupil> shortPupil = NirCamTemplateFieldFactory.makeShortPupilField(this);
    private final CosiConstrainedSelection<NirCamPupil> longPupil = NirCamTemplateFieldFactory.makeLongPupilField(this);

    private final CosiConstrainedSelection<NirCamReadPattern> readoutPatternField = NirCamTemplateFieldFactory.makeReadoutPatternField
(this);
    private final CosiConstrainedInt numberOfGroupsField = NirCamTemplateFieldFactory.makeNumberOfGroupsField(this);
    private final CosiConstrainedInt numberOfIntegrationsField = NirCamTemplateFieldFactory.makeNumberOfIntsField(this);
    private final CosiConstrainedDouble actualExpTimeField = NirCamTemplateFieldFactory.makeActualExpTimeField(this);

    private final CosiConstrainedInt steps1 = NirCamTemplateFieldFactory.makeLinearActuatorStepsField(this, START_STEPS+" 1");
    private final CosiConstrainedInt steps2 = NirCamTemplateFieldFactory.makeLinearActuatorStepsField(this, START_STEPS+" 2");
    private final CosiConstrainedInt steps3 = NirCamTemplateFieldFactory.makeLinearActuatorStepsField(this, START_STEPS+" 3");

    private final CosiConstrainedInt sensor1 = NirCamTemplateFieldFactory.makeLinearActuatorSensorUnitsField(this, START_SENSOR_UNITS
+" 1");
    private final CosiConstrainedInt sensor2 = NirCamTemplateFieldFactory.makeLinearActuatorSensorUnitsField(this, START_SENSOR_UNITS
+" 2");
    private final CosiConstrainedInt sensor3 = NirCamTemplateFieldFactory.makeLinearActuatorSensorUnitsField(this, START_SENSOR_UNITS
+" 3");

    private final CosiConstrainedInt phase1 = NirCamTemplateFieldFactory.makeLinearActuatorMotorPhaseField(this, START_MOTOR_PHASE+"
1");
    private final CosiConstrainedInt phase2 = NirCamTemplateFieldFactory.makeLinearActuatorMotorPhaseField(this, START_MOTOR_PHASE+"
2");
    private final CosiConstrainedInt phase3 = NirCamTemplateFieldFactory.makeLinearActuatorMotorPhaseField(this, START_MOTOR_PHASE+"
3");

    private final NirCamLinearActuatorTable laTable = NirCamTemplateFieldFactory.makeNirCamLinearActuatorTableField(this);

    private final CosiBooleanField returnToStart = NirCamTemplateFieldFactory.makeFocusReturnToStartField(this);

    //Tina Groups
    private final TinaFieldGroup filterGroup = new TinaFieldGroup(this, NirCamTemplateFieldFactory.FILTER);
    private final TinaFieldGroup pupilGroup = new TinaFieldGroup(this, NirCamTemplateFieldFactory.PUPIL);
    private final TinaFieldGroup expTimeGroup = new TinaFieldGroup(this, EXPOSURE_TIME);
    private final TinaFieldGroup startStepsGroup = new TinaFieldGroup(this, START_STEPS);
    private final TinaFieldGroup startSensorGroup = new TinaFieldGroup(this, START_SENSOR_UNITS);
    private final TinaFieldGroup startPhaseGroup = new TinaFieldGroup(this, START_MOTOR_PHASE);
    {
        //Filter Group
```

```
        shortFilter.setTinaFieldGroup(filterGroup);
        filterGroup.setLabel(shortFilter, NirCamTemplateFieldFactory.SHORT_FILTER);
        longFilter.setTinaFieldGroup(filterGroup);
        filterGroup.setLabel(longFilter, NirCamTemplateFieldFactory.LONG_FILTER);

        //Pupil Group
        shortPupil.setTinaFieldGroup(pupilGroup);
        pupilGroup.setLabel(shortPupil, NirCamTemplateFieldFactory.SHORT_PUPIL);
        longPupil.setTinaFieldGroup(pupilGroup);
        pupilGroup.setLabel(longPupil, NirCamTemplateFieldFactory.LONG_PUPIL);

        //Exp Time Group
        readoutPatternField.setTinaFieldGroup(expTimeGroup);
        expTimeGroup.setLabel(readoutPatternField,NirCamTemplateFieldFactory.READOUT_PATTERN);
        numberOfGroupsField.setTinaFieldGroup(expTimeGroup);
        expTimeGroup.setLabel(numberOfGroupsField,NirCamTemplateFieldFactory.NUMBER_OF_GROUPS);
        numberOfIntegrationsField.setTinaFieldGroup(expTimeGroup);
        expTimeGroup.setLabel(numberOfIntegrationsField,NirCamTemplateFieldFactory.NUMBER_OF_INTS);
        actualExpTimeField.setTinaFieldGroup(expTimeGroup);
        expTimeGroup.setLabel(actualExpTimeField,NirCamTemplateFieldFactory.ACTUAL_EXP_TIME);

        //Start Steps
        steps1.setTinaFieldGroup(startStepsGroup);
        startStepsGroup.setLabel(steps1, LINEAR_ACTUATOR+" 1");
        steps2.setTinaFieldGroup(startStepsGroup);
        startStepsGroup.setLabel(steps2, LINEAR_ACTUATOR+" 2");
        steps3.setTinaFieldGroup(startStepsGroup);
        startStepsGroup.setLabel(steps3, LINEAR_ACTUATOR+" 3");

        //Start Sensor Units
        sensor1.setTinaFieldGroup(startSensorGroup);
//        startSensorGroup.setLabel(sensor1, LINEAR_ACTUATOR+1);
        sensor2.setTinaFieldGroup(startSensorGroup);
//        startSensorGroup.setLabel(sensor2, LINEAR_ACTUATOR+2);
        sensor3.setTinaFieldGroup(startSensorGroup);
//        startSensorGroup.setLabel(sensor3, LINEAR_ACTUATOR+3);

        //Start Motor Phase
        phase1.setTinaFieldGroup(startPhaseGroup);
//        startPhaseGroup.setLabel(phase1, LINEAR_ACTUATOR+1);
        phase2.setTinaFieldGroup(startPhaseGroup);
//        startPhaseGroup.setLabel(phase2, LINEAR_ACTUATOR+2);
        phase3.setTinaFieldGroup(startPhaseGroup);
//        startPhaseGroup.setLabel(phase3, LINEAR_ACTUATOR+3);

        templateProperties = new TinaField[] {module, shortFilter, longFilter, shortPupil, longPupil, readoutPatternField,
                numberOfGroupsField, numberOfIntegrationsField, actualExpTimeField,
                steps1, steps2, steps3,
                sensor1, sensor2, sensor3,
                phase1, phase2, phase3,
```

```
            laTable, returnToStart};
}

//--------------------Constructors--------------------
public NirCamFocusTemplate () {
    super(NirCamInstrument.getInstance());
    Cosi.completeInitialization(this, NirCamFocusTemplate.class);
}

public NirCamFilter getShortFilter() {
    return shortFilter.get();
}

public void setShortFilter(NirCamFilter iFilter) {
    shortFilter.set(iFilter);
}

public NirCamFilter getLongFilter() {
    return longFilter.get();
}

public void setLongFilter(NirCamFilter iFilter) {
    longFilter.set(iFilter);
}

public NirCamPupil getShortPupil() {
    return shortPupil.get();
}

public void setShortPupil(NirCamPupil iPupil) {
    shortPupil.set(iPupil);
}

public NirCamPupil getLongPupil() {
    return longPupil.get();
}

public void setLongPupil(NirCamPupil iPupil) {
    longPupil.set(iPupil);
}

public NirCamReadPattern getReadoutPattern() {
    return readoutPatternField.get();
}

public void setReadoutPattern(NirCamReadPattern iPattern) {
    readoutPatternField.set(iPattern);
}

public Integer getNumberOfGroups() {
```

```
    return numberOfGroupsField.get();
}

public void setNumberOfGroups(Integer i) {
    numberOfGroupsField.set(i);
}

public Integer getNumberOfIntegrations() {
    return numberOfIntegrationsField.get();
}

public void setNumberOfIntegrations(Integer i) {
    numberOfIntegrationsField.set(i);
}

public NirCamModule getModule() {
    return module.get();
}

public void setModule(NirCamModule iModule) {
    module.set(iModule);
}

public Double getActualExpTime() {
    return actualExpTimeField.get();
}

public Integer getStartSteps(int iWhich) {
    switch (iWhich) {
        case 1: return steps1.get();
        case 2: return steps2.get();
        case 3: return steps3.get();
        default:
            throw new IllegalArgumentException();
    }
}

public void setStartSteps(int iWhich, Integer iValue) {
    switch (iWhich) {
        case 1: steps1.set(iValue); break;
        case 2: steps2.set(iValue); break;
        case 3: steps3.set(iValue); break;
        default:
            throw new IllegalArgumentException();
    }
}

public Integer getStartSensorUnits(int iWhich) {
    switch (iWhich) {
        case 1: return sensor1.get();
```

```
            case 2: return sensor2.get();
            case 3: return sensor3.get();
            default:
                throw new IllegalArgumentException();
        }
    }

    public void setStartSensorUnits(int iWhich, Integer iValue) {
        switch (iWhich) {
            case 1: sensor1.set(iValue); break;
            case 2: sensor2.set(iValue); break;
            case 3: sensor3.set(iValue); break;
            default:
                throw new IllegalArgumentException();
        }
    }

    public Integer getStartMotorPhase(int iWhich) {
        switch (iWhich) {
            case 1: return phase1.get();
            case 2: return phase2.get();
            case 3: return phase3.get();
            default:
                throw new IllegalArgumentException();
        }
    }

    public void setStartMotorPhase(int iWhich, Integer iValue) {
        switch (iWhich) {
            case 1: phase1.set(iValue); break;
            case 2: phase2.set(iValue); break;
            case 3: phase3.set(iValue); break;
            default:
                throw new IllegalArgumentException();
        }
    }

    public List<NirCamLinearActuatorTableRow> getLAPositions () {
        return laTable.getValue();
    }

    public void addLAPosition(NirCamLinearActuatorTableRow iRow) {
        laTable.addField(iRow);
    }

    public Boolean getReturnToStart () {
        return returnToStart.get();
    }

    public void setReturnToStart (Boolean iValue) {
```

```
        returnToStart.set(iValue);
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import java.security.InvalidParameterException;

/**
 * NirCamLinearActuatorTable - provides a table interface for defining a list of focus
 * positions that will be executed for the associated observation.
 *
 * @author Rob Douglas
 */
public class NirCamLinearActuatorTable extends AbstractTinaMultiField<NirCamLinearActuatorTableRow> {
    private static final int MAX_POSITIONS = 10;
    private static final String MAX_POSITIONS_DIAG = "No more than "+MAX_POSITIONS+" positions may be used.";

    //--------------------Constructors--------------------
    public NirCamLinearActuatorTable (TinaDocumentElement iContainer, String iName) {
        super(iContainer, iName);
        Cosi.completeInitialization(this, NirCamLinearActuatorTable.class);
    }

    //--------------------Accessors--------------------
    public int getColumnCount () {
        return 3;
    }

    public Class<?> getColumnClass (int iColumn) {
        switch (iColumn) {
            case 0: return CosiConstrainedInt.class;
            case 1: return CosiConstrainedInt.class;
            case 2: return CosiConstrainedInt.class;
            default:
                throw new InvalidParameterException("Expected column less than " + getColumnCount());
        }
    }

    // BE ADVISED!!! There is some inconsistent behavior in the order properties are fired
    // between ConstrainedInt and ConstrainedSelection.  This means that when this method is
    // called, ConstrainedSelection still has its old, original value, but ConstrainedInt
    // has been updated with its new value!
    public void setValueAt (Object iValue, int iRow, int iColumn) {
        NirCamLinearActuatorTableRow lField = getValue().get(iRow);

        if ((iValue != null) && (!"".equals(iValue.toString()))) {
            switch (iColumn) {
                case 0: lField.setLAPosition(1, iValue.toString()); break;
                case 1: lField.setLAPosition(2, iValue.toString()); break;
                case 2: lField.setLAPosition(3, iValue.toString()); break;
                default:
                    throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
```

```
        }
    }
}

public String getColumnName(int iColumn) {
    switch (iColumn) {
        case 0: return NirCamLinearActuatorTableRow.ABSOLUTE_POSITION+" 1";
        case 1: return NirCamLinearActuatorTableRow.ABSOLUTE_POSITION+" 2";
        case 2: return NirCamLinearActuatorTableRow.ABSOLUTE_POSITION+" 3";
        default:
            throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
    }
}

public Object getValueAt(int rowIndex, int iColumn) {
    NirCamLinearActuatorTableRow lField = getValue().get(rowIndex);
    switch (iColumn) {
        case 0: return lField.la1Position;
        case 1: return lField.la2Position;
        case 2: return lField.la3Position;
    }
    throw new InvalidParameterException("Col count was "+iColumn+".  Expected column less than " + getColumnCount());
}

public NirCamFocusTemplate getTemplate () {
    return (NirCamFocusTemplate)getContainer();
}

//--------------------Methods--------------------
public void configureDefaultEditors (JTable iTable) {
    iTable.setDefaultEditor(CosiConstrainedInt.class, new DefaultTinaCosiFieldEditor());
}

public boolean areRowsValid() {
    for(NirCamLinearActuatorTableRow lRow : getValue()) {
        if(lRow.isRowValid() == false) {
            return false;
        }
    }
    return true;
}

@Override
public int addField(NirCamLinearActuatorTableRow iField) {
    int lRetVal = super.addField(iField);
    getTemplate().add(iField, false);
    return lRetVal;
}

@Override
```

```
    protected NirCamLinearActuatorTableRow newField() {
        NirCamLinearActuatorTableRow lRow = new NirCamLinearActuatorTableRow();
        getTemplate().add(lRow, false);
        return lRow;
    }

    //--------------------Constraints--------------------
    @CosiConstraint
    private void cosiCheckMaxPositions () {
        int lNumRows = getRowCount();
        DiagnosticManager.ensureDiagnostic(this, this, this,
                Diagnostic.ERROR,
                MAX_POSITIONS_DIAG,
                MAX_POSITIONS_DIAG,
                lNumRows>MAX_POSITIONS);
    }
}
```

```
 * This code was developed at the Space Telescope Science Institute under contract
package edu.stsci.jwst.apt.model.template.nircam;

import org.jdom.Element;

/**
 * NirCamLinearActuatorTableRow - represents a single set of positions for the linear
 * actuators in a NirCam Focus
 *
 * @author Rob Douglas
 */
public class NirCamLinearActuatorTableRow extends AbstractTinaDocumentElement
implements TinaMultiFieldField {
    //--------------------Statics--------------------
    public static final String ABSOLUTE_POSITION = "Absolute Position in Steps";
    //--------------------Fields--------------------
    protected final CosiConstrainedInt la1Position = NirCamTemplateFieldFactory.makeLinearActuatorStepsField(this, ABSOLUTE_POSITION+"
1");
    protected final CosiConstrainedInt la2Position = NirCamTemplateFieldFactory.makeLinearActuatorStepsField(this, ABSOLUTE_POSITION+"
2");
    protected final CosiConstrainedInt la3Position = NirCamTemplateFieldFactory.makeLinearActuatorStepsField(this, ABSOLUTE_POSITION+"
3");

    //--------------------Constructors--------------------
    public NirCamLinearActuatorTableRow () {
        super();
        Cosi.completeInitialization(this,NirCamLinearActuatorTableRow.class);
    }

    //--------------------Accessors--------------------
    public NirCamFocusTemplate getTemplate() {
        return (NirCamFocusTemplate)getParent();
    }

    public Integer getLAPosition(int iWhich) {
        switch (iWhich) {
            case 1: return la1Position.get();
            case 2: return la2Position.get();
            case 3: return la3Position.get();
            default:
                throw new IllegalArgumentException();
        }
    }

    public void setLAPosition(int iWhich, Integer iValue) {
        switch (iWhich) {
            case 1: la1Position.set(iValue); break;
            case 2: la2Position.set(iValue); break;
            case 3: la3Position.set(iValue); break;
            default:
```

```
                throw new IllegalArgumentException();
        }
    }

    public void setLAPosition(int iWhich, String iValue) {
        switch (iWhich) {
            case 1: la1Position.setValueFromString(iValue); break;
            case 2: la2Position.setValueFromString(iValue); break;
            case 3: la3Position.setValueFromString(iValue); break;
            default:
                throw new IllegalArgumentException();
        }
    }

    public boolean isRowValid() {
        return (!la1Position.isRequired() || la1Position.isSpecified()) &&
        (!la2Position.isRequired() || la2Position.isSpecified()) &&
        (!la3Position.isRequired() || la3Position.isSpecified()) &&
        !la1Position.isOutOfRange() &&
        !la2Position.isOutOfRange() &&
        !la3Position.isOutOfRange();
    }

    @Override
    public String getTypeName() {
        // TODO Auto-generated method stub
        return null;
    }

    public Element getDomElement() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

```java
package edu.stsci.jwst.apt.model.template.nircam;

import edu.stsci.CoSI.Cosi;

public class NirCamWheelThresholdCurrentTemplate extends JwstTemplate<NirCamInstrument> {
    //--------------------Fields--------------------
    {
        templateProperties = new TinaField[] {};
    }

    //--------------------Constructors--------------------
    public NirCamWheelThresholdCurrentTemplate () {
        super(NirCamInstrument.getInstance());
        Cosi.completeInitialization(this, NirCamWheelThresholdCurrentTemplate.class);
    }

}
```

```java
package edu.stsci.jwst.apt.io;

import java.io.BufferedWriter;

public class JwstProposalFile {

    //[start]--------------------Statics----------------------
    private static ObjectFactory fFactory = new ObjectFactory();
    private static final String PROPERTY_VERSION        = "VERSION";
    private static final String PROPERTY_SCHEMA_VERSION = "SCHEMA_VERSION";
    private static final String PROPERTY_DATE           = "DATE";
    private static final String PROPERTY_PLATFORM       = "PLATFORM";
    private static final String PROPERTY_USER           = "USER";
    private static final String PROPERTY_PHASE          = "PHASE";

    private static final String JWST_DM_SCHEMA_FILE = "JwstDMSchema.xsd";
    private static int fSchemaVersion = 0;
    private static Schema DM_SCHEMA;

    //[end]

    //[start]--------------------Fields----------------------
    private File fFile = null;
    private JAXBContext JAXB_CONTEXT;
    private boolean fValidationErrorDetected = false;
    private boolean fCommentWritten = false;

    private DefaultValidationEventHandler fValidationErrorHandler = new DefaultValidationEventHandler() {
        @Override
        public boolean handleEvent(ValidationEvent arg0) {
            if(super.handleEvent(arg0) == false) {
                fValidationErrorDetected = true;
            }
            return true;
        }
    };
    //[end]

    //[start]--------------------Initializers-----------------
    static {
        try {
            URL lSchemaURL = JwstProposalFile.class.getResource("/" + JWST_DM_SCHEMA_FILE);
            Source lSchemaSource = new StreamSource(lSchemaURL.toExternalForm());

            ArrayList<Source> lTemplateSources = findTemplateSchemas();
            lTemplateSources.add(lSchemaSource);

            DM_SCHEMA = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI).newSchema(lTemplateSources.toArray(new Source
[0]));
```

```
        DocumentBuilderFactory lFactory = DocumentBuilderFactory.newInstance();
        lFactory.setNamespaceAware(true);
        DocumentBuilder lDocBuilder = lFactory.newDocumentBuilder();
        Document lJwstSchema = lDocBuilder.parse(JwstProposalFile.class.getResource("/" + JWST_DM_SCHEMA_FILE).toExternalForm());
        fSchemaVersion = Integer.parseInt(lJwstSchema.getDocumentElement().getAttribute("version"));
        MessageLogger.getInstance().writeDebug(null, "Schema version: ["+fSchemaVersion+"]");
    } catch(Exception ex) {
        ex.printStackTrace();
    }
}

{
    try {
        JAXB_CONTEXT = JAXBContext.newInstance("edu.stsci.jwst.apt.jaxb", JwstProposalFile.class.getClassLoader());
    } catch (JAXBException e) {
        e.printStackTrace();
    }
}
//[end]

//[start]--------------------Constructors-----------------
public JwstProposalFile(File iFile) {
    fFile = iFile;
}
//[end]

//[start]--------------------Methods----------------------

//   [start] Common Methods
// This method looks through the JwstDMJaxb JAR file and finds all schema files.  Once found,
// a Source object is created.  Note that only the "TemplateSchemas" directory and all
// subdirectories are searched.
private static ArrayList<Source> findTemplateSchemas() {

    try {
        ArrayList<Source> lTemplateSources = new ArrayList<Source>();

        String jarFile = null;
        String pathSeparator = System.getProperty("path.separator");
        for (String s : System.getProperty("java.class.path").split(pathSeparator)) {
            if (s.contains("JwstDmJaxb")) {
                jarFile = s;
                break;
            }
        }

        JarFile lJarFile = new JarFile(jarFile);
        Enumeration<JarEntry> lJarEntries = lJarFile.entries();
        while(lJarEntries.hasMoreElements()) {
            JarEntry lEntry = lJarEntries.nextElement();
```

```
                if(lEntry.getName().startsWith("TemplateSchemas") && lEntry.getName().endsWith(".xsd")) {
                    lTemplateSources.add(
                            new StreamSource(
                                    JwstProposalFile.class.getResource("/" + lEntry.getName()).toExternalForm()));
                }
            }
            return lTemplateSources;
        }
        catch(Exception ex) {
            ex.printStackTrace();
            return null;
        }
    }

    /**
     *
     * @param iProposal - the JWST Proposal to be written
     * @return RETURN_CODE to indicate the state of the operation
     *              - SUCCESS if the save completed successfully with validation
     *              - SUCCESS_WITH_ISSUE if the save completed successfully, but only after disabling validation
     *              - EXCEPTION if an exception was internally caught and handled
     */
    public FILE_RETURN_CODE save(JwstProposalSpecification iProposal) {

        try {
            JaxbJwstProposal lProp = convertToJaxb(iProposal);
            lProp.setSchemaVersion(fSchemaVersion);

            // Write the file
            final BufferedWriter lWriter = new BufferedWriter(new FileWriter(fFile));
            Marshaller lMarshaller = JAXB_CONTEXT.createMarshaller();
            lMarshaller.setSchema(DM_SCHEMA);
            lMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

            fValidationErrorDetected = false;
            lMarshaller.setEventHandler(fValidationErrorHandler);

            lMarshaller.setListener(new Listener() {
                @Override
                public void beforeMarshal(Object source) {
                    super.beforeMarshal(source);
                    try {
                        // We need this goofy "fCommentWritten" variable because the "beforeMarhsal" method
                        // gets called twice on the JWST Proposal element.  Seems like a bug in Jaxb.
                        if (fCommentWritten == false && source instanceof JaxbJwstProposal) {
                            lWriter.write("<!--
********************************************************************************************************************* -->"
                                    + System.getProperty("line.separator"));
                            lWriter.write("<!-- This file is automatically generated and should not be edited by hand. Editing this
file directly is at your own risk. -->"
```

```
                                    + System.getProperty("line.separator"));
                            lWriter.write("<!--
********************************************************************************************************************** -->"
                                    + System.getProperty("line.separator"));
                            fCommentWritten = true;
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            });

            fCommentWritten = false;
            lMarshaller.marshal(lProp, lWriter);

            FILE_RETURN_CODE lRetCode = FILE_RETURN_CODE.RET_SUCCESS;

            if(fValidationErrorDetected) {
                lRetCode = FILE_RETURN_CODE.RET_SUCCESS_WITH_ISSUE;
            }

            lWriter.close();

            return lRetCode;
        }
        catch(Exception ex) {
            ex.printStackTrace();
            return FILE_RETURN_CODE.RET_EXCEPTION;
        }
    }

    /**
     *
     * @param iFile - The XML file containing the JWST Proposal
     * @param oProposal - The object into which the XML file is unmarshalled
     * @return RETURN_CODE to indicate the state of the operation
     *              - SUCCESS if the load completed successfully with validation
     *              - SUCCESS_WITH_ISSUE if the load completed successfully, but only after disabling validation
     *              - EXCEPTION if an exception was internally caught and handled
     */
    public FILE_RETURN_CODE load(JwstProposalSpecification oProposal) {
        Document lConvertedDoc = null;
        try {
            Unmarshaller lUnmarshaller = JAXB_CONTEXT.createUnmarshaller();
            lUnmarshaller.setSchema(DM_SCHEMA);

            lConvertedDoc = JwstProposalFileConverter.convertFile(fFile, fSchemaVersion);
            if(lConvertedDoc == null) {
                throw new Exception("File conversion failed!");
            }
```

```
        FILE_RETURN_CODE lRetCode = FILE_RETURN_CODE.RET_SUCCESS;
        fValidationErrorDetected = false;
        lUnmarshaller.setEventHandler(fValidationErrorHandler);

        JaxbJwstProposal lProp = (JaxbJwstProposal)lUnmarshaller.unmarshal(lConvertedDoc);
        if(fValidationErrorDetected) {
            lRetCode = FILE_RETURN_CODE.RET_SUCCESS_WITH_ISSUE;
        }

        convertToDm(lProp, oProposal);
        return lRetCode;
    }
    catch (Exception ex) {
        ex.printStackTrace();
        return FILE_RETURN_CODE.RET_EXCEPTION;
    }
}

private static Properties createMetadataRecord(JwstProposalSpecification iProp) {
    Properties metadata = new Properties();
    addProperty(metadata, PROPERTY_VERSION, AptController.getApplicationVersionString());
    addProperty(metadata, PROPERTY_SCHEMA_VERSION, Integer.toString(fSchemaVersion));
    addProperty(metadata, PROPERTY_DATE, Calendar.getInstance().getTime().toString());
    addProperty(metadata, PROPERTY_PLATFORM, System.getProperty("os.name") + " - " + System.getProperty("os.arch"));
    addProperty(metadata, PROPERTY_USER, System.getProperty("user.name"));
    addProperty(metadata, PROPERTY_PHASE, iProp.getProposalPhase());

    return metadata;
}

//
// Private methods used to save/load parts of the proposal
//
// DEVELOPER NOTES:
// ===============
// The current methodology is to reconstruct the tree from top down.  This is to ensure that
// little things in Tina like "getDocument" and "elementInsertedIntoHierarchy" always have a
// complete ancestry (e.g. getDocument never returns null)
//
// To ensure this, the convertToDm methods follow the following paradigm:
//   - Create a JWST object
//   - Attach the JWST object to its parent
//   - Extract the contents of the JAXB object into the JWST object
//
// The general signature of the methods below are as follows:
//   - There are a pair of methods for each conversion.  The first method is
//
//         <Jaxb Type> convertToJaxb( <DM Type> )
//
```

```
//     This method takes a Document Model element and converts it into its Jaxb counterpart.
//     The second method's signature is
//
//         void convertToDm( <Jaxb Type, input>, <DM Type, output> )
//
//     Note this method does not return an object, but instead populates the second argument.  This
//     is due to the methodology listed above, i.e. we need to construct the object, attach it to
//     its parent, then populate its contents.

public static JaxbJwstProposal convertToJaxb(JwstProposalSpecification iProp) {
    JaxbJwstProposal lProp = fFactory.createJaxbJwstProposal();

    List<Properties> lMetadata = iProp.getMetadata();
    lMetadata.add(createMetadataRecord(iProp));
    lProp.setProposalHistory(convertToJaxb(lMetadata.toArray(new Properties[0])));

    lProp.setProposalInformation(convertToJaxb(iProp.getPropInfo()));
    lProp.setTargets(convertToJaxb(iProp.getTargets()));
    lProp.setDataRequests(convertToJaxb(iProp.getDataRequestFolder()));

    return lProp;
}

public static void convertToDm(JaxbJwstProposal iJaxbProp, JwstProposalSpecification oDmProp) {

    convertToDm(iJaxbProp.getProposalHistory(), oDmProp);
    convertToDm(iJaxbProp.getProposalInformation(), oDmProp.getPropInfo());
    convertToDm(iJaxbProp.getTargets(), oDmProp.getTargets());
    convertToDm(iJaxbProp.getDataRequests(), oDmProp.getDataRequestFolder());
}

public static JaxbProposalHistoryType convertToJaxb(Properties[] iRecords) {
    JaxbProposalHistoryType lHistory = new JaxbProposalHistoryType();
    for(Properties record : iRecords) {
        JaxbHistoryEntryType lJaxbRecord = new JaxbHistoryEntryType();

        lJaxbRecord.setAPTVersion(record.getProperty(PROPERTY_VERSION));
        if (record.getProperty(PROPERTY_SCHEMA_VERSION) != null) {
            lJaxbRecord.setSchemaVersion(Integer.parseInt(record.getProperty(PROPERTY_SCHEMA_VERSION)));
        }
        lJaxbRecord.setDate(record.getProperty(PROPERTY_DATE));
        lJaxbRecord.setPlatform(record.getProperty(PROPERTY_PLATFORM));
        lJaxbRecord.setUser(record.getProperty(PROPERTY_USER));
        lJaxbRecord.setProposalPhase(record.getProperty(PROPERTY_PHASE));

        lHistory.getHistoryEntry().add(lJaxbRecord);
    }

    return lHistory;
}
```

```
public static void convertToDm(JaxbProposalHistoryType iJaxbHistory, JwstProposalSpecification oDmProp) {
    if(iJaxbHistory != null) {
        for(JaxbHistoryEntryType lJaxbHistoryEntry : iJaxbHistory.getHistoryEntry()) {
            Properties props = new Properties();
            addProperty(props, PROPERTY_VERSION, lJaxbHistoryEntry.getAPTVersion());
            if (lJaxbHistoryEntry.getSchemaVersion() != null) {
                addProperty(props, PROPERTY_SCHEMA_VERSION, Integer.toString(lJaxbHistoryEntry.getSchemaVersion()));
            }
            addProperty(props, PROPERTY_DATE, lJaxbHistoryEntry.getDate());
            addProperty(props, PROPERTY_PLATFORM, lJaxbHistoryEntry.getPlatform());
            addProperty(props, PROPERTY_USER, lJaxbHistoryEntry.getUser());
            addProperty(props, PROPERTY_PHASE, lJaxbHistoryEntry.getProposalPhase());

            oDmProp.getMetadata().add(props);
        }
    }
}
private static void addProperty(Properties iProps, String iPropName, String iPropValue) {
    if(iPropValue != null) {
        iProps.setProperty(iPropName, iPropValue);
    }
}
//[end]

//   [start] Common Proposal Elements
public static JaxbProposalInformation convertToJaxb(JwstProposalInformation iPropInfo) {
    JaxbProposalInformation lInfo = fFactory.createJaxbProposalInformation();
    lInfo.setTitle(iPropInfo.getTitle());
    lInfo.setAbstract(iPropInfo.getAbstract());
    lInfo.setProposalID(iPropInfo.getProposalID());
    lInfo.setProposalCategory(convertToJaxb(iPropInfo.getCategory()));
    lInfo.setProposalCategorySubtype(convertToJaxb(iPropInfo.getSubCategory()));

    lInfo.setProposalCategory(convertToJaxb(iPropInfo.getCategory()));
    lInfo.setPrincipalInvestigator(convertToJaxb((JwstPrincipalInvestigator)iPropInfo.getPrincipalInvestigator()));
    lInfo.getCoInvestigator().addAll(convertToJaxb(iPropInfo.getCoInvestigators()));

    return lInfo;
}

public static void convertToDm(JaxbProposalInformation iPropInfo, JwstProposalInformation oPropInfo) {
    oPropInfo.setTitle(iPropInfo.getTitle());
    oPropInfo.setAbstract(iPropInfo.getAbstract());
    oPropInfo.setProposalID(iPropInfo.getProposalID());
    oPropInfo.setCategory(convertToDm(iPropInfo.getProposalCategory()));
    oPropInfo.setSubCategory(convertToDm(iPropInfo.getProposalCategorySubtype()));

    JwstPrincipalInvestigator lPI = new JwstPrincipalInvestigator();
    oPropInfo.setPrincipalInvestigator(lPI);
```

```
        convertToDm(iPropInfo.getPrincipalInvestigator(), lPI);

        for(JaxbCoInvestigator lJaxbCoI : iPropInfo.getCoInvestigator()) {
            JwstCoInvestigator lCoI = new JwstCoInvestigator();
            oPropInfo.addCoInvestigator(lCoI);
            convertToDm(lJaxbCoI, lCoI);
        }
    }

    public static JaxbTargets convertToJaxb(JwstTargets iTargs) {
        JaxbTargets lTargs = fFactory.createJaxbTargets();
        for(JwstFixedTarget lTarg : iTargs.getChildren(JwstFixedTarget.class)) {
            lTargs.getTarget().add(convertToJaxb(lTarg));
        }

        return lTargs;
    }

    public static void convertToDm(JaxbTargets iTargs, JwstTargets oTargs) {
        for(JaxbAbstractTargetType lJaxbTarget : iTargs.getTarget()) {
            if(lJaxbTarget instanceof JaxbFixedTargetType) {
                JwstFixedTarget lFixedTarget = new JwstFixedTarget();
                oTargs.add(lFixedTarget, true);
                convertToDm((JaxbFixedTargetType)lJaxbTarget, lFixedTarget);
            }
        }
    }

    public static JaxbFixedTargetType convertToJaxb(JwstFixedTarget iTarg) {
        JaxbFixedTargetType lTarg = fFactory.createJaxbFixedTargetType();
        lTarg.setAnnualParallax(iTarg.getAnnualParallaxAsString());
        lTarg.setComments(iTarg.getComment());
        lTarg.setDecProperMotion(iTarg.getDecPMString());
        lTarg.setEpoch(iTarg.getEpochString());
        lTarg.setNumber(iTarg.getNumber());
        lTarg.setRAProperMotion(iTarg.getRAPMString());
        lTarg.setRedshift(iTarg.getRedshiftAsString());
        lTarg.setTargetID(iTarg.getName());
        lTarg.setTargetName(iTarg.getName());

        JaxbEquatorialCoordinatesType lCoords = convertToJaxb(iTarg.getCoordinates());
        JaxbRAUncertaintyType lRaUnc = fFactory.createJaxbRAUncertaintyType();
        JaxbDecUncertaintyType lDecUnc = fFactory.createJaxbDecUncertaintyType();

        //Pat: Finish these
//      lTarg.setDescription(iTarg.getDescription());
//      lRaUnc.setValue(iTarg.getRaUncArcsec());
//      lRaUnc.setUnits("Arcseconds");
//      lDecUnc.setValue(iTarg.getDecUncArcsec());
//      lDecUnc.setUnits("Arcseconds");
```

```
        lCoords.setRAUncertainty(lRaUnc);
        lCoords.setDecUncertainty(lDecUnc);

        lTarg.setEquatorialCoordinates(lCoords);


        JaxbFluxesType lFluxes = new JaxbFluxesType();
        for (AcqFluxTableEntry lEntry : iTarg.getAcqFluxes()) {
            JaxbFluxType lFlux = new JaxbFluxType();

            lFlux.setInstrument(convertToJaxb(lEntry.getInstrument()));
            lFlux.setAcqFilter(lEntry.getFilterAsString());
            lFlux.setFluxValue(lEntry.getFlux());

            lFluxes.getFlux().add(lFlux);
        }
        lTarg.setFluxes(lFluxes);

        return lTarg;
    }

    public static void convertToDm(JaxbFixedTargetType iTarget, JwstFixedTarget oTarget) {
        oTarget.setAnnualParallax(iTarget.getAnnualParallax());
        oTarget.setComment(iTarget.getComments());
        oTarget.setDecPM(iTarget.getDecProperMotion());

        oTarget.setEpoch(iTarget.getEpoch());

        Coordinates lCoords = new Coordinates();
        oTarget.setCoordinates(lCoords);
        convertToDm(iTarget.getEquatorialCoordinates(), lCoords);

        //Pat: Finish these
//      oTarget.setDescription(iTarget.getDescription());
        oTarget.setNumber(iTarget.getNumber());
        oTarget.setRAPM(iTarget.getRAProperMotion());
        oTarget.setRedshift(iTarget.getRedshift());
//      oTarget.setTargetId(iTarget.getTargetID());
        oTarget.setName(iTarget.getTargetName());

        if (iTarget.getFluxes() != null) {
            for (JaxbFluxType lFlux : iTarget.getFluxes().getFlux()) {
                AcqFluxTableEntry lEntry = new AcqFluxTableEntry();
                oTarget.addFlux(lEntry);

                lEntry.setInstrument(convertToDm(lFlux.getInstrument()));
                lEntry.setFilterFromString(lFlux.getAcqFilter());
                lEntry.setFlux(lFlux.getFluxValue());
            }
```

```
    }
}

public static JaxbEquatorialCoordinatesType convertToJaxb(Coordinates iCoords) {
    JaxbEquatorialCoordinatesType lCoords = fFactory.createJaxbEquatorialCoordinatesType();

    if(iCoords != null && iCoords.raToString() != null) {
        JaxbRAType lRaType = fFactory.createJaxbRAType();
        StringTokenizer lTokenizer = new StringTokenizer(iCoords.raToString(Coordinates.COLON_SEPARATOR_STYLE),
                String.valueOf(Coordinates.COLON_SEPARATOR));
        lRaType.setHours(Integer.parseInt(lTokenizer.nextToken().replace("+", "")));
        lRaType.setMinutes(Integer.parseInt(lTokenizer.nextToken()));
        lRaType.setSeconds(lTokenizer.nextToken());
        lCoords.setRA(lRaType);
    }

    if(iCoords != null && iCoords.decToString() != null) {
        JaxbDecType lDecType = fFactory.createJaxbDecType();
        StringTokenizer lTokenizer = new StringTokenizer(iCoords.decToString(Coordinates.COLON_SEPARATOR_STYLE),
                String.valueOf(Coordinates.COLON_SEPARATOR));
        lDecType.setDegrees(Integer.parseInt(lTokenizer.nextToken().replace("+", "")));
        lDecType.setMinutes(Integer.parseInt(lTokenizer.nextToken()));
        lDecType.setSeconds(lTokenizer.nextToken());
        lCoords.setDec(lDecType);
    }

    return lCoords;
}

public static void convertToDm(JaxbEquatorialCoordinatesType iCoords, Coordinates oCoords) {
    String lRaString = null;
    String lDecString = null;

    if(iCoords.getRA() != null) {
        lRaString = iCoords.getRA().getHours() + ":" + iCoords.getRA().getMinutes() + ":" + iCoords.getRA().getSeconds();
    }
    if(iCoords.getDec() != null) {
        lDecString = iCoords.getDec().getDegrees() + ":" + iCoords.getDec().getMinutes() + ":" + iCoords.getDec().getSeconds();
    }

    if(lRaString != null && lDecString != null) {
        Coordinates lTemp = Coordinates.valueOf(Coordinates.COLON_SEPARATOR_STYLE, lRaString, lDecString);
        oCoords.setRa(lTemp.getRa());
        oCoords.setDec(lTemp.getDec());
    }
}

public static JaxbDataRequestsType convertToJaxb(JwstDataRequestFolder iFolder) {
    JaxbDataRequestsType lRequests = new JaxbDataRequestsType();
    for(JwstObservationGroup lObsGroup : iFolder.getChildren(JwstObservationGroup.class)) {
```

```
            lRequests.getObservationGroup().add(convertToJaxb(lObsGroup));
        }
        return lRequests;
    }

    public static void convertToDm(JaxbDataRequestsType iJaxbFolder, JwstDataRequestFolder oFolder) {
        if(iJaxbFolder != null) {
            for(JaxbObservationGroupType lJaxbObsGroup : iJaxbFolder.getObservationGroup()) {
                JwstObservationGroup lGroup = new JwstObservationGroup();
                oFolder.add(lGroup, true);
                convertToDm(lJaxbObsGroup, lGroup);
            }
        }
    }

    public static JaxbObservationGroupType convertToJaxb(JwstObservationGroup iObsGroup) {
        JaxbObservationGroupType lJaxbGroup = new JaxbObservationGroupType();
        lJaxbGroup.setLabel(iObsGroup.getObsGroupLabel());
        lJaxbGroup.setComments(iObsGroup.getComments());

        for(JwstObservation lObs : iObsGroup.getChildren(JwstObservation.class)) {
            lJaxbGroup.getObservation().add(convertToJaxb(lObs));
        }

        return lJaxbGroup;
    }

    public static void convertToDm(JaxbObservationGroupType iJaxbObsGroup, JwstObservationGroup oObsGroup) {
        oObsGroup.setObsGroupLabel(iJaxbObsGroup.getLabel());
        oObsGroup.setComments(iJaxbObsGroup.getComments());

        for(JaxbObservationType lJaxbObs : iJaxbObsGroup.getObservation()) {
            JwstObservation lObs = new JwstObservation();
            oObsGroup.add(lObs, true);
            convertToDm(lJaxbObs, lObs);
        }
    }

    public static JaxbObservationType convertToJaxb(JwstObservation iObs) {
        JaxbObservationType lJaxbObs = new JaxbObservationType();
        lJaxbObs.setNumber(iObs.getNumber());

        lJaxbObs.setLabel(iObs.getLabel());
        lJaxbObs.setComments(iObs.getComments());

        Target lTarget = iObs.getTarget();
        if(lTarget != null) {
            lJaxbObs.setTargetID(iObs.getTargetAsString());
        }
```

```
        lJaxbObs.setInstrument(convertToJaxb(iObs.getInstrument()));
        lJaxbObs.setTemplate(convertToJaxb(iObs.getTemplate()));

        return lJaxbObs;
    }

    public static void convertToDm(JaxbObservationType iJaxbObs, JwstObservation oObs) {
        oObs.setNumber(iJaxbObs.getNumber());
        oObs.setLabel(iJaxbObs.getLabel());
        oObs.setComments(iJaxbObs.getComments());
        oObs.setTarget(iJaxbObs.getTargetID());
        oObs.setInstrument(convertToDm(iJaxbObs.getInstrument()));

        // Note: This call creates a new Template object in the Observation.  We need to get that
        // object and populate it rather than just creating our own Template object so CoSI stuff
        // doesn't break.
        oObs.setTemplateChooser(getTemplateFactory(iJaxbObs.getTemplate()));
        convertToDm(iJaxbObs.getTemplate(), oObs.getTemplate());
    }

    public static JaxbPrincipalInvestigator convertToJaxb(JwstPrincipalInvestigator iPI) {
        JaxbPrincipalInvestigator lPI = fFactory.createJaxbPrincipalInvestigator();
        lPI.setInvestigatorAddress(convertToJaxbAddress(iPI));

        return lPI;
    }

    public static void convertToDm(JaxbPrincipalInvestigator iPI, JwstPrincipalInvestigator oPI) {
        oPI.setESAMember(iPI.getInvestigatorAddress().isESAMember());
        oPI.setFirstName(iPI.getInvestigatorAddress().getFirstName());
        oPI.setHonorific(iPI.getInvestigatorAddress().getHonorific());
        oPI.setLastName(iPI.getInvestigatorAddress().getLastName());
        oPI.setMiddleInitial(iPI.getInvestigatorAddress().getMiddleInitial());
        oPI.setSuffix(iPI.getInvestigatorAddress().getSuffix());

        oPI.getAddress().setCountry(iPI.getInvestigatorAddress().getCountry());
        oPI.getAddress().setEMail(iPI.getInvestigatorAddress().getEMail());
        oPI.getAddress().setInstitution(iPI.getInvestigatorAddress().getInstitution());
        oPI.getAddress().setUSState(iPI.getInvestigatorAddress().getUSState());
    }

    public static List<JaxbCoInvestigator> convertToJaxb(List<CoInvestigator> iCoIs) {
        Vector<JaxbCoInvestigator> lCoIs = new Vector<JaxbCoInvestigator>();

        for(CoInvestigator lJwstCoI : iCoIs) {
            lCoIs.add(convertToJaxb(lJwstCoI));
        }

        return lCoIs;
    }
```

```
public static JaxbCoInvestigator convertToJaxb(CoInvestigator iCoI) {
    JaxbCoInvestigator lCoI = fFactory.createJaxbCoInvestigator();
    lCoI.setAdminUSPI(iCoI.getAdminUSPI());
    lCoI.setContact(iCoI.getContact());
    lCoI.setInvestigatorAddress(convertToJaxbAddress(iCoI));

    return lCoI;
}

public static void convertToDm(JaxbCoInvestigator iCoI, JwstCoInvestigator oCoI) {
    oCoI.setAdminUSPI(iCoI.isAdminUSPI());
    oCoI.setContact(iCoI.isContact());

    oCoI.setESAMember(iCoI.getInvestigatorAddress().isESAMember());
    oCoI.setFirstName(iCoI.getInvestigatorAddress().getFirstName());
    oCoI.setHonorific(iCoI.getInvestigatorAddress().getHonorific());
    oCoI.setLastName(iCoI.getInvestigatorAddress().getLastName());
    oCoI.setMiddleInitial(iCoI.getInvestigatorAddress().getMiddleInitial());
    oCoI.setSuffix(iCoI.getInvestigatorAddress().getSuffix());

    oCoI.getAddress().setCountry(iCoI.getInvestigatorAddress().getCountry());
    oCoI.getAddress().setEMail(iCoI.getInvestigatorAddress().getEMail());
    oCoI.getAddress().setInstitution(iCoI.getInvestigatorAddress().getInstitution());
    oCoI.getAddress().setUSState(iCoI.getInvestigatorAddress().getUSState());
}

public static JaxbInvestigatorAddressType convertToJaxbAddress(Investigator iPI) {
    JaxbInvestigatorAddressType lAddr = fFactory.createJaxbInvestigatorAddressType();

    lAddr.setCountry(iPI.getAddress().getCountry());
    lAddr.setEMail(iPI.getAddress().getEMail());
    lAddr.setESAMember(iPI.getESAMember());
    lAddr.setFirstName(iPI.getFirstName());
    lAddr.setHonorific(iPI.getHonorific());
    lAddr.setInstitution(iPI.getAddress().getInstitutionName());
    lAddr.setLastName(iPI.getLastName());
    lAddr.setMiddleInitial(iPI.getMiddleInitial());
    lAddr.setSuffix(iPI.getSuffix());
    lAddr.setUSState(iPI.getAddress().getStateField().getName());

    return lAddr;
}

public static JaxbProposalCategoryType convertToJaxb(Category iCategory) {
    JaxbProposalCategoryType lType = new JaxbProposalCategoryType();

    if(Category.AR == iCategory) {
        lType.setAR(new JaxbARType());
    }
```

```
        else if(Category.CAL == iCategory) {
            lType.setCAL(new JaxbCALType());
        }
        else if(Category.ENG == iCategory) {
            lType.setENG(new JaxbENGType());
        }
        else if(Category.GO == iCategory) {
            lType.setGO(new JaxbGOType());
        }
        else if(Category.GODD == iCategory) {
            lType.setGODD(new JaxbGODDType());
        }
        else if(Category.GTO == iCategory) {
            lType.setGTO(new JaxbGTOType());
        }
        else if(Category.NASA == iCategory) {
            lType.setNASA(new JaxbNASAType());
        }
        return lType;
    }

    public static Category convertToDm(JaxbProposalCategoryType iCategory) {
        if(iCategory != null) {
            if(iCategory.getAR() != null) {
                return Category.AR;
            }
            if(iCategory.getCAL() != null) {
                return Category.CAL;
            }
            if(iCategory.getENG() != null) {
                return Category.ENG;
            }
            if(iCategory.getGO() != null) {
                return Category.GO;
            }
            if(iCategory.getGODD() != null) {
                return Category.GODD;
            }
            if(iCategory.getGTO() != null) {
                return Category.GTO;
            }
            if(iCategory.getNASA() != null) {
                return Category.NASA;
            }
        }

        return null;
    }

    public static JaxbProposalCategorySubtypeType convertToJaxb(SubCategory iSubCategory) {
```

```
        if(iSubCategory != null) {
            if(iSubCategory == SubCategory.FGS) {
                return JaxbProposalCategorySubtypeType.FGS;
            }
            if(iSubCategory == SubCategory.MIRI) {
                return JaxbProposalCategorySubtypeType.MIRI;
            }
            if(iSubCategory == SubCategory.NIRCAM) {
                return JaxbProposalCategorySubtypeType.NIRCAM;
            }
            if(iSubCategory == SubCategory.NIRSPEC) {
                return JaxbProposalCategorySubtypeType.NIRSPEC;
            }
            if(iSubCategory == SubCategory.SC) {
                return JaxbProposalCategorySubtypeType.SC;
            }
            if(iSubCategory == SubCategory.TFI) {
                return JaxbProposalCategorySubtypeType.TFI;
            }
        }

        return null;
    }

    public static SubCategory convertToDm(JaxbProposalCategorySubtypeType iCategory) {
        if(iCategory != null) {
            if(iCategory == JaxbProposalCategorySubtypeType.FGS) {
                return SubCategory.FGS;
            }
            if(iCategory == JaxbProposalCategorySubtypeType.MIRI) {
                return SubCategory.MIRI;
            }
            if(iCategory == JaxbProposalCategorySubtypeType.NIRCAM) {
                return SubCategory.NIRCAM;
            }
            if(iCategory == JaxbProposalCategorySubtypeType.NIRSPEC) {
                return SubCategory.NIRSPEC;
            }
            if(iCategory == JaxbProposalCategorySubtypeType.SC) {
                return SubCategory.SC;
            }
            if(iCategory == JaxbProposalCategorySubtypeType.TFI) {
                return SubCategory.TFI;
            }
        }

        return null;
    }

    public static JaxbInstrumentType convertToJaxb(JwstInstrument iInstrument) {
```

```
    if(iInstrument != null) {
        if(iInstrument instanceof MiriInstrument) {
            return JaxbInstrumentType.MIRI;
        }
        if(iInstrument instanceof NirCamInstrument) {
            return JaxbInstrumentType.NIRCAM;
        }
        if(iInstrument instanceof NirSpecInstrument) {
            return JaxbInstrumentType.NIRSPEC;
        }
        if(iInstrument instanceof TfiInstrument) {
            return JaxbInstrumentType.TFI;
        }
    }

    return null;
}

public static JwstInstrument convertToDm(JaxbInstrumentType iInstrument) {
    if(iInstrument != null) {
        switch(iInstrument) {
            case TFI:
            case FGS:
                return TfiInstrument.getInstance();
            case MIRI:
                return MiriInstrument.getInstance();
            case NIRCAM:
                return NirCamInstrument.getInstance();
            case NIRSPEC:
                return NirSpecInstrument.getInstance();
        }
    }
    return null;
}
//[end]

//   [start] Template Conversions
public static JaxbTemplateType convertToJaxb(JwstTemplate<? extends JwstInstrument> iTemplate) {
    JaxbTemplateType lJaxbTemplate = new JaxbTemplateType();
    if (iTemplate instanceof MiriImagingTemplate) {
        lJaxbTemplate.setMiriImaging(convertToJaxb((MiriImagingTemplate)iTemplate));
    }
    else if (iTemplate instanceof MiriAnnealTemplate) {
        lJaxbTemplate.setMiriAnneal(convertToJaxb((MiriAnnealTemplate)iTemplate));
    }
    else if (iTemplate instanceof MiriMrsFlatTemplate) {
        lJaxbTemplate.setMiriMRSFlat(convertToJaxb((MiriMrsFlatTemplate)iTemplate));
    }
    else if (iTemplate instanceof MiriDarkTemplate) {
        lJaxbTemplate.setMiriDark(convertToJaxb((MiriDarkTemplate)iTemplate));
```

```
    }
    else if (iTemplate instanceof MiriImagingFlatTemplate) {
        lJaxbTemplate.setMiriImagingFlat(convertToJaxb((MiriImagingFlatTemplate)iTemplate));
    }
    else if (iTemplate instanceof MiriLrsTemplate) {
        lJaxbTemplate.setMiriLRS(convertToJaxb((MiriLrsTemplate)iTemplate));
    }
    else if (iTemplate instanceof MiriMrsTemplate) {
        lJaxbTemplate.setMiriMRS(convertToJaxb((MiriMrsTemplate)iTemplate));
    }
    else if (iTemplate instanceof MiriCoronTemplate) {
        lJaxbTemplate.setMiriCoron(convertToJaxb((MiriCoronTemplate)iTemplate));
    }
    else if (iTemplate instanceof NirCamImagingTemplate) {
        lJaxbTemplate.setNircamImaging(convertToJaxb((NirCamImagingTemplate)iTemplate));
    }
    else if (iTemplate instanceof NirCamCoronTemplate) {
        lJaxbTemplate.setNircamCoron(convertToJaxb((NirCamCoronTemplate)iTemplate));
    }
    else if (iTemplate instanceof NirCamDarkTemplate) {
        lJaxbTemplate.setNircamDark(convertToJaxb((NirCamDarkTemplate)iTemplate));
    }
    else if (iTemplate instanceof NirCamFlatTemplate) {
        lJaxbTemplate.setNircamInternalFlat(convertToJaxb((NirCamFlatTemplate)iTemplate));
    }
    else if (iTemplate instanceof NirCamWheelExerciseTemplate) {
        lJaxbTemplate.setNircamWheelExercise(convertToJaxb((NirCamWheelExerciseTemplate)iTemplate));
    }
    else if (iTemplate instanceof NirCamFocusTemplate) {
        lJaxbTemplate.setNircamFocus(convertToJaxb((NirCamFocusTemplate)iTemplate));
    }
    else if (iTemplate instanceof NirCamWheelThresholdCurrentTemplate) {
        lJaxbTemplate.setNircamWheelThresholdCurrent(convertToJaxb((NirCamWheelThresholdCurrentTemplate)iTemplate));
    }

    return lJaxbTemplate;
}

// Developer's Note:
// ================
// This method breaks the model outlined above.  The reason is that the JwstTemplate
// is not a TinaDocumentElement CHILD of the Observation, so we don't need to enforce
// construction of the tree in a top-down order.  We also can't instantiate the
// JwstTemplate object prior to calling this method because it is an abstract class.
public static void convertToDm(JaxbTemplateType iJaxbTemplate, JwstTemplate<? extends JwstInstrument> oDmTemplate) {
    if (iJaxbTemplate == null) {
        return;
    }

    if (iJaxbTemplate.getMiriImaging() != null) {
```

```
            convertToDm(iJaxbTemplate.getMiriImaging(), (MiriImagingTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getMiriAnneal() != null) {
            convertToDm(iJaxbTemplate.getMiriAnneal(), (MiriAnnealTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getMiriMRSFlat() != null) {
            convertToDm(iJaxbTemplate.getMiriMRSFlat(), (MiriMrsFlatTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getMiriDark() != null) {
            convertToDm(iJaxbTemplate.getMiriDark(), (MiriDarkTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getMiriImagingFlat() != null) {
            convertToDm(iJaxbTemplate.getMiriImagingFlat(), (MiriImagingFlatTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getMiriMRS() != null) {
            convertToDm(iJaxbTemplate.getMiriMRS(), (MiriMrsTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getMiriLRS() != null) {
            convertToDm(iJaxbTemplate.getMiriLRS(), (MiriLrsTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getMiriCoron() != null) {
            convertToDm(iJaxbTemplate.getMiriCoron(), (MiriCoronTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getNircamImaging() != null) {
            convertToDm(iJaxbTemplate.getNircamImaging(), (NirCamImagingTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getNircamCoron() != null) {
            convertToDm(iJaxbTemplate.getNircamCoron(), (NirCamCoronTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getNircamDark() != null) {
            convertToDm(iJaxbTemplate.getNircamDark(), (NirCamDarkTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getNircamInternalFlat() != null) {
            convertToDm(iJaxbTemplate.getNircamInternalFlat(), (NirCamFlatTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getNircamWheelExercise() != null) {
            convertToDm(iJaxbTemplate.getNircamWheelExercise(), (NirCamWheelExerciseTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getNircamFocus() != null) {
            convertToDm(iJaxbTemplate.getNircamFocus(), (NirCamFocusTemplate)oDmTemplate);
        }
        else if (iJaxbTemplate.getNircamWheelThresholdCurrent() != null) {
            convertToDm(iJaxbTemplate.getNircamWheelThresholdCurrent(), (NirCamWheelThresholdCurrentTemplate)oDmTemplate);
        }
    }

    private static JwstTemplateFactory getTemplateFactory(JaxbTemplateType iType) {
        if (iType.getMiriImaging() != null) {
            return JwstTemplateFactory.MIRI_IMAGING;
        }
```

```
    else if (iType.getMiriAnneal() != null) {
        return JwstTemplateFactory.MIRI_ANNEAL;
    }
    else if (iType.getMiriMRSFlat() != null) {
        return JwstTemplateFactory.MIRI_MRS_FLAT;
    }
    else if (iType.getMiriDark() != null) {
        return JwstTemplateFactory.MIRI_DARK;
    }
    else if (iType.getMiriImagingFlat() != null) {
        return JwstTemplateFactory.MIRI_IMG_FLAT;
    }
    else if (iType.getMiriMRS() != null) {
        return JwstTemplateFactory.MIRI_MRS;
    }
    else if (iType.getMiriLRS() != null) {
        return JwstTemplateFactory.MIRI_LRS;
    }
    else if (iType.getMiriCoron() != null) {
        return JwstTemplateFactory.MIRI_CORON;
    }


    else if (iType.getNircamImaging() != null) {
        return JwstTemplateFactory.NIRCAM_IMAGING;
    }
    else if (iType.getNircamCoron() != null) {
        return JwstTemplateFactory.NIRCAM_CORON;
    }
    else if (iType.getNircamDark() != null) {
        return JwstTemplateFactory.NIRCAM_DARK;
    }
    else if (iType.getNircamInternalFlat() != null) {
        return JwstTemplateFactory.NIRCAM_FLAT;
    }
    else if (iType.getNircamWheelExercise() != null) {
        return JwstTemplateFactory.NIRCAM_WHEEL_EXERCISE;
    }
    else if (iType.getNircamFocus() != null) {
        return JwstTemplateFactory.NIRCAM_FOCUS;
    }
    else if (iType.getNircamWheelThresholdCurrent() != null) {
        return JwstTemplateFactory.NIRCAM_WHEEL_THRESHOLD_CURRENT;
    }


    return null;
}

//[end]
```

```
    //   [start] Miri Templates
    public static JaxbMiriImaging convertToJaxb(MiriImagingTemplate iTemplate) {
        JaxbMiriImaging lJaxbTemplate = new JaxbMiriImaging();

        lJaxbTemplate.setObjectType(MiriObjectType_JaxbObjectTypeType.getValueFromKey(iTemplate.getObjectType()));
        lJaxbTemplate.setSubarray(MiriSubarray_JaxbSubarrayType.getValueFromKey(iTemplate.getSubarray()));

        JaxbFiltersType lJaxbFilters = new JaxbFiltersType();
        for(MiriImagingExposureSpecTableRow lRow : iTemplate.getExposures()) {
            JaxbFilterConfigType lJaxbFilter = new JaxbFilterConfigType();

//            lJaxbFilter.setActualExposureTime(lRow.getTotalIntegrationTime());
            lJaxbFilter.setFilter(MiriFilter_JaxbFilterType.getValueFromKey(lRow.getFilter()));
            lJaxbFilter.setGroups(lRow.getNumberOfGroups());
            lJaxbFilter.setIntegrations(lRow.getNumberOfIntegrations());
            lJaxbFilter.setReadoutPattern(MiriReadPattern_JaxbReadoutPatternType.getValueFromKey(lRow.getReadoutPattern()));
            lJaxbFilter.setRequestedExposureTime(lRow.getRequestedExpTime());

            lJaxbFilters.getFilterConfig().add(lJaxbFilter);
        }
        lJaxbTemplate.setFilters(lJaxbFilters);

        return lJaxbTemplate;
    }

    public static void convertToDm(JaxbMiriImaging iJaxbTemplate, MiriImagingTemplate oTemplate) {
        oTemplate.setObjectType(MiriObjectType_JaxbObjectTypeType.getKeyFromValue(iJaxbTemplate.getObjectType()));
        oTemplate.setSubarray(MiriSubarray_JaxbSubarrayType.getKeyFromValue(iJaxbTemplate.getSubarray()));

        for(JaxbFilterConfigType lJaxbFilter : iJaxbTemplate.getFilters().getFilterConfig()) {
            MiriImagingExposureSpecTableRow lRow = new MiriImagingExposureSpecTableRow(oTemplate.getExposureTable());
            oTemplate.addExposure(lRow);

            lRow.setFilter(MiriFilter_JaxbFilterType.getKeyFromValue(lJaxbFilter.getFilter()));
            lRow.setRequestedExpTime(lJaxbFilter.getRequestedExposureTime());

            // Developer's Note:
            // We will not read in the derived parameters.  The MIRI Imaging Calculator will compute
            // these values.  We need to consider situations when we want to read in these parameters
            // rather than recompute them.
            //
            // Arguably, if we are not reading them, we should not be saving them.
            //
//            lRow.setCalculatedExpTime(lJaxbFilter.getActualExposureTime());
//            lRow.setNumberOfGroups(lJaxbFilter.getGroups());
//            lRow.setNumberOfIntegrations(lJaxbFilter.getIntegrations());
//            lRow.setReadoutPattern(convertToDm(lJaxbFilter.getReadoutPattern()));
        }
    }
```

```
public static JaxbMiriAnneal convertToJaxb(MiriAnnealTemplate iTemplate) {
    JaxbMiriAnneal lJaxbTemplate = new JaxbMiriAnneal();

    lJaxbTemplate.setDetector(MiriDetector_JaxbDetectorType.getValueFromKey(iTemplate.getDetector()));

    return lJaxbTemplate;
}

public static void convertToDm(JaxbMiriAnneal iJaxbTemplate, MiriAnnealTemplate oTemplate) {
    oTemplate.setDetector(MiriDetector_JaxbDetectorType.getKeyFromValue(iJaxbTemplate.getDetector()));
}

public static JaxbMiriMRSFlat convertToJaxb(MiriMrsFlatTemplate iTemplate) {
    JaxbMiriMRSFlat lJaxbTemplate = new JaxbMiriMRSFlat();

    lJaxbTemplate.setFlatSuite(MiriFlatSuite_JaxbFlatSuiteType.getValueFromKey(iTemplate.getFlatSuite()));
    lJaxbTemplate.setWavelength(MiriWavelength_JaxbWavelengthType.getValueFromKey(iTemplate.getWavelength()));
    lJaxbTemplate.setNumberOfIntegrations(iTemplate.getNumberOfIntegrations());

    return lJaxbTemplate;
}

public static void convertToDm(JaxbMiriMRSFlat iJaxbTemplate, MiriMrsFlatTemplate oTemplate) {
    oTemplate.setFlatSuite(MiriFlatSuite_JaxbFlatSuiteType.getKeyFromValue(iJaxbTemplate.getFlatSuite()));
    oTemplate.setWavelength(MiriWavelength_JaxbWavelengthType.getKeyFromValue(iJaxbTemplate.getWavelength()));
    oTemplate.setNumberOfIntegrations(iJaxbTemplate.getNumberOfIntegrations());
}

public static JaxbMiriDark convertToJaxb(MiriDarkTemplate iTemplate) {
    JaxbMiriDark lJaxbTemplate = new JaxbMiriDark();

    lJaxbTemplate.setDetector(MiriDetector_JaxbDetectorType.getValueFromKey(iTemplate.getDetector()));

    edu.stsci.jwst.apt.template.miridark.JaxbFiltersType lJaxbFilters =
        new edu.stsci.jwst.apt.template.miridark.JaxbFiltersType();

    for(MiriDarkExposureSpecTableRow lRow : iTemplate.getExposures()) {
        edu.stsci.jwst.apt.template.miridark.JaxbFilterConfigType lJaxbFilter
            = new edu.stsci.jwst.apt.template.miridark.JaxbFilterConfigType();

        lJaxbFilter.setNumberOfExposures(lRow.getNumberOfExposures());
        lJaxbFilter.setReadoutPattern(MiriReadPattern_JaxbReadoutPatternType.getValueFromKey(lRow.getReadoutPattern()));
        lJaxbFilter.setGroups(lRow.getNumberOfGroups());
        lJaxbFilter.setIntegrations(lRow.getNumberOfIntegrations());

        lJaxbFilters.getFilterConfig().add(lJaxbFilter);
    }
    lJaxbTemplate.setFilters(lJaxbFilters);
```

```
    return lJaxbTemplate;
}

public static void convertToDm(JaxbMiriDark iJaxbTemplate, MiriDarkTemplate oTemplate) {
    oTemplate.setDetector(MiriDetector_JaxbDetectorType.getKeyFromValue(iJaxbTemplate.getDetector()));

    for(edu.stsci.jwst.apt.template.miridark.JaxbFilterConfigType lJaxbFilter :
            iJaxbTemplate.getFilters().getFilterConfig()) {

        MiriDarkExposureSpecTableRow lRow = new MiriDarkExposureSpecTableRow(oTemplate.getExposureTable());
        lRow.setNumberOfExposures(lJaxbFilter.getNumberOfExposures());
        lRow.setReadoutPattern(MiriReadPattern_JaxbReadoutPatternType.getKeyFromValue(lJaxbFilter.getReadoutPattern()));
        lRow.setNumberOfGroups(lJaxbFilter.getGroups());
        lRow.setNumberOfIntegrations(lJaxbFilter.getIntegrations());

        oTemplate.addExposure(lRow);
    }
}

public static JaxbMiriImagingFlat convertToJaxb(MiriImagingFlatTemplate iTemplate) {
    JaxbMiriImagingFlat lJaxbTemplate = new JaxbMiriImagingFlat();

    lJaxbTemplate.setFilter(MiriFilter_JaxbFilterType.getValueFromKey(iTemplate.getFilter()));
    lJaxbTemplate.setFlatSuite(MiriFlatSuite_JaxbFlatSuiteType.getValueFromKey(iTemplate.getFlatSuite()));
    lJaxbTemplate.setNumberOfIntegrations(iTemplate.getNumInts());

    return lJaxbTemplate;
}

public static void convertToDm(JaxbMiriImagingFlat iJaxbTemplate, MiriImagingFlatTemplate oTemplate) {
    oTemplate.setFilter(MiriFilter_JaxbFilterType.getKeyFromValue(iJaxbTemplate.getFilter()));
    oTemplate.setFlatSuite(MiriFlatSuite_JaxbFlatSuiteType.getKeyFromValue(iJaxbTemplate.getFlatSuite()));
    oTemplate.setNumInts(iJaxbTemplate.getNumberOfIntegrations());
}

public static JaxbMiriMRS convertToJaxb(MiriMrsTemplate iTemplate) {
    JaxbMiriMRS lJaxbTemplate = new JaxbMiriMRS();

    if(iTemplate.getAcqTarget() != null && iTemplate.getAcqTargetAsString() != null) {
        lJaxbTemplate.setAcqTargetID(iTemplate.getAcqTargetAsString());
    }

    lJaxbTemplate.setAcqFilter(MiriFilter_JaxbFilterType.getValueFromKey(iTemplate.getAcqFilter()));
    lJaxbTemplate.setObjectType(MiriObjectType_JaxbObjectTypeType.getValueFromKey(iTemplate.getObjectType()));
    lJaxbTemplate.setWavelength(MiriWavelength_JaxbWavelengthType.getValueFromKey(iTemplate.getWavelength()));

    lJaxbTemplate.setRequestedExposureTime(iTemplate.getRequestedExpTime());
    lJaxbTemplate.setReadoutPattern(MiriReadPattern_JaxbReadoutPatternType.getValueFromKey(iTemplate.getReadoutPattern()));
    lJaxbTemplate.setNumberOfGroups(iTemplate.getNumberOfGroups());
    lJaxbTemplate.setNumberOfIntegrations(iTemplate.getNumberOfIntegrations());
```

```
//        lJaxbTemplate.setCalculatedExposureTime()

        return lJaxbTemplate;
    }

    public static void convertToDm(JaxbMiriMRS iJaxbTemplate, MiriMrsTemplate oTemplate) {
        oTemplate.setAcqTargetFromString(iJaxbTemplate.getAcqTargetID());
        oTemplate.setAcqFilter(MiriFilter_JaxbFilterType.getKeyFromValue(iJaxbTemplate.getAcqFilter()));
        oTemplate.setObjectType(MiriObjectType_JaxbObjectTypeType.getKeyFromValue(iJaxbTemplate.getObjectType()));
        oTemplate.setWavelength(MiriWavelength_JaxbWavelengthType.getKeyFromValue(iJaxbTemplate.getWavelength()));

        oTemplate.setRequestedExpTime(iJaxbTemplate.getRequestedExposureTime());
        oTemplate.setReadoutPattern(MiriReadPattern_JaxbReadoutPatternType.getKeyFromValue(iJaxbTemplate.getReadoutPattern()));
        oTemplate.setNumberOfGroups(iJaxbTemplate.getNumberOfGroups());
        oTemplate.setNumberOfIntegrations(iJaxbTemplate.getNumberOfIntegrations());
//        oTemplate.setCalculatedExposureTime

    }

    public static JaxbMiriLRS convertToJaxb(MiriLrsTemplate iTemplate) {
        JaxbMiriLRS lJaxbTemplate = new JaxbMiriLRS();

        if(iTemplate.getAcqTarget() != null && iTemplate.getAcqTargetAsString() != null) {
            lJaxbTemplate.setAcqTargetID(iTemplate.getAcqTargetAsString());
        }
        lJaxbTemplate.setAcqFilter(MiriFilter_JaxbFilterType.getValueFromKey(iTemplate.getAcqFilter()));
        lJaxbTemplate.setObjectType(MiriObjectType_JaxbObjectTypeType.getValueFromKey(iTemplate.getObjectType()));
        lJaxbTemplate.setRequestedExposureTime(iTemplate.getRequestedExpTime());
        lJaxbTemplate.setReadoutPattern(MiriReadPattern_JaxbReadoutPatternType.getValueFromKey(iTemplate.getReadoutPattern()));
        lJaxbTemplate.setNumberOfGroups(iTemplate.getNumberOfGroups());
        lJaxbTemplate.setNumberOfIntegrations(iTemplate.getNumberOfIntegrations());
//        lJaxbTemplate.setCalculatedExposureTime();

        return lJaxbTemplate;
    }

    public static void convertToDm(JaxbMiriLRS iJaxbTemplate, MiriLrsTemplate oTemplate) {
        oTemplate.setAcqTargetFromString(iJaxbTemplate.getAcqTargetID());
        oTemplate.setAcqFilter(MiriFilter_JaxbFilterType.getKeyFromValue(iJaxbTemplate.getAcqFilter()));
        oTemplate.setObjectType(MiriObjectType_JaxbObjectTypeType.getKeyFromValue(iJaxbTemplate.getObjectType()));
        oTemplate.setRequestedExpTime(iJaxbTemplate.getRequestedExposureTime());
        oTemplate.setReadoutPattern(MiriReadPattern_JaxbReadoutPatternType.getKeyFromValue(iJaxbTemplate.getReadoutPattern()));
        oTemplate.setNumberOfGroups(iJaxbTemplate.getNumberOfGroups());
        oTemplate.setNumberOfIntegrations(iJaxbTemplate.getNumberOfIntegrations());
//        oTemplate.setCalculatedExposureTime
    }

    public static JaxbMiriCoron convertToJaxb(MiriCoronTemplate iTemplate) {
        JaxbMiriCoron lJaxbTemplate = new JaxbMiriCoron();
```

```
        if(iTemplate.getAcqTarget() != null && iTemplate.getAcqTargetAsString() != null) {
            lJaxbTemplate.setAcqTargetID(iTemplate.getAcqTargetAsString());
        }
        lJaxbTemplate.setAcqFilter(MiriFilter_JaxbFilterType.getValueFromKey(iTemplate.getAcqFilter()));
        lJaxbTemplate.setObjectType(MiriObjectType_JaxbObjectTypeType.getValueFromKey(iTemplate.getObjectType()));


        edu.stsci.jwst.apt.template.miricoron.JaxbFiltersType lJaxbFilters =
            new edu.stsci.jwst.apt.template.miricoron.JaxbFiltersType();

        for(MiriCoronExposureSpecTableRow lRow : iTemplate.getFilters()) {
            edu.stsci.jwst.apt.template.miricoron.JaxbFilterConfigType lJaxbFilter
                = new edu.stsci.jwst.apt.template.miricoron.JaxbFilterConfigType();

            lJaxbFilter.setMask(MiriMask_JaxbMaskType.getValueFromKey(lRow.getMask()));
            lJaxbFilter.setFilter(MiriFilter_JaxbCoronFilterType.getValueFromKey(lRow.getCoronFilter()));
            lJaxbFilter.setRequestedExposureTime(lRow.getRequestedExpTime());
            lJaxbFilter.setReadoutPattern(MiriReadPattern_JaxbReadoutPatternType.getValueFromKey(lRow.getReadoutPattern()));
            lJaxbFilter.setNumberOfGroups(lRow.getNumberOfGroups());
            lJaxbFilter.setNumberOfIntegrations(lRow.getNumberOfIntegrations());
//            lJaxbFilter.setCalculatedExposureTime();

            lJaxbFilters.getFilterConfig().add(lJaxbFilter);
        }
        lJaxbTemplate.setFilters(lJaxbFilters);



        return lJaxbTemplate;
    }

    public static void convertToDm(JaxbMiriCoron iJaxbTemplate, MiriCoronTemplate oTemplate) {
//        oTemplate.setAcqTargetID
        oTemplate.setAcqFilter(MiriFilter_JaxbFilterType.getKeyFromValue(iJaxbTemplate.getAcqFilter()));
        oTemplate.setObjectType(MiriObjectType_JaxbObjectTypeType.getKeyFromValue(iJaxbTemplate.getObjectType()));

        for(edu.stsci.jwst.apt.template.miricoron.JaxbFilterConfigType lJaxbFilter :
            iJaxbTemplate.getFilters().getFilterConfig()) {

            MiriCoronExposureSpecTableRow lRow = new MiriCoronExposureSpecTableRow(oTemplate.getFilterTable());
            lRow.setCoronFilter(MiriFilter_JaxbCoronFilterType.getKeyFromValue(lJaxbFilter.getFilter()));
            lRow.setRequestedExpTime(lJaxbFilter.getRequestedExposureTime());
            lRow.setReadoutPattern(MiriReadPattern_JaxbReadoutPatternType.getKeyFromValue(lJaxbFilter.getReadoutPattern()));
            lRow.setNumberOfGroups(lJaxbFilter.getNumberOfGroups());
            lRow.setNumberOfIntegrations(lJaxbFilter.getNumberOfIntegrations());
//            lRow.setCalculatedExposureTime

            oTemplate.addExposure(lRow);
        }
    }
```

```
//[end]

//   [start] Nircam Templates
public static JaxbNircamImaging convertToJaxb(NirCamImagingTemplate iTemplate) {
    JaxbNircamImaging lJaxbTemplate = new JaxbNircamImaging();

    lJaxbTemplate.setModule(NirCamModule_JaxbModuleType.getValueFromKey(iTemplate.getModule()));
    lJaxbTemplate.setReadoutRegion(NirCamSubarray_JaxbReadoutRegionType.getValueFromKey(iTemplate.getSubarray()));

    edu.stsci.jwst.apt.template.nircamimaging.JaxbFiltersType lJaxbFilters
        = new edu.stsci.jwst.apt.template.nircamimaging.JaxbFiltersType();

    for (NirCamImagingExposureSpecTableRow lRow : iTemplate.getExposures()) {
        edu.stsci.jwst.apt.template.nircamimaging.JaxbFilterConfigType lJaxbFilter
        = new edu.stsci.jwst.apt.template.nircamimaging.JaxbFilterConfigType();

        lJaxbFilter.setActualExposureTime(lRow.getActualExpTime());
        lJaxbFilter.setGroups(lRow.getNumberOfGroups());
        lJaxbFilter.setIntegrations(lRow.getNumberOfIntegrations());
        lJaxbFilter.setLongFilter(NirCamFilter_JaxbFilterType.getValueFromKey(lRow.getLongFilter()));
        lJaxbFilter.setReadoutPattern(NirCamReadPattern_JaxbReadoutPatternType.getValueFromKey(lRow.getReadoutPattern()));
        lJaxbFilter.setRequestedExposureTime(lRow.getRequestedExpTime());
        lJaxbFilter.setShortFilter(NirCamFilter_JaxbFilterType.getValueFromKey(lRow.getShortFilter()));

        lJaxbFilters.getFilterConfig().add(lJaxbFilter);
    }
    lJaxbTemplate.setFilters(lJaxbFilters);

    return lJaxbTemplate;
}

public static void convertToDm(JaxbNircamImaging iJaxbTemplate, NirCamImagingTemplate oTemplate) {
    oTemplate.setModule(NirCamModule_JaxbModuleType.getKeyFromValue(iJaxbTemplate.getModule()));
    oTemplate.setSubarray(NirCamSubarray_JaxbReadoutRegionType.getKeyFromValue(iJaxbTemplate.getReadoutRegion()));

    for (edu.stsci.jwst.apt.template.nircamimaging.JaxbFilterConfigType lJaxbFilter :
        iJaxbTemplate.getFilters().getFilterConfig()) {
        NirCamImagingExposureSpecTableRow lRow = new NirCamImagingExposureSpecTableRow();

        lRow.setRequestedExpTime(lJaxbFilter.getRequestedExposureTime());
        lRow.setNumberOfGroups(lJaxbFilter.getGroups());
        lRow.setNumberOfIntegrations(lJaxbFilter.getIntegrations());
        lRow.setLongFilter(NirCamFilter_JaxbFilterType.getKeyFromValue(lJaxbFilter.getLongFilter()));
        lRow.setShortFilter(NirCamFilter_JaxbFilterType.getKeyFromValue(lJaxbFilter.getShortFilter()));
        lRow.setReadoutPattern(NirCamReadPattern_JaxbReadoutPatternType.getKeyFromValue(lJaxbFilter.getReadoutPattern()));

        oTemplate.addExposure(lRow);
    }
}
```

```
public static JaxbNircamCoron convertToJaxb(NirCamCoronTemplate iTemplate) {
    JaxbNircamCoron lJaxbTemplate = new JaxbNircamCoron();

    lJaxbTemplate.setAcqTargetID(iTemplate.getAcqTargetAsString());
    lJaxbTemplate.setAcqFilter(NirCamFilter_JaxbFilterType.getValueFromKey(iTemplate.getAcqFilter()));
    lJaxbTemplate.setCoronMask(NirCamMask_JaxbNircamMask.getValueFromKey(iTemplate.getMask()));

    edu.stsci.jwst.apt.template.nircamcoron.JaxbFiltersType lJaxbFilters
        = new edu.stsci.jwst.apt.template.nircamcoron.JaxbFiltersType();

    for (NirCamCoronExposureSpecTableRow lRow : iTemplate.getExposures()) {
        edu.stsci.jwst.apt.template.nircamcoron.JaxbFilterConfigType lJaxbFilter =
            new edu.stsci.jwst.apt.template.nircamcoron.JaxbFilterConfigType();
        lJaxbFilter.setFilter(NirCamFilter_JaxbFilterType.getValueFromKey(lRow.getFilter()));
        lJaxbFilter.setNumberOfGroups(lRow.getNumberOfGroups());
        lJaxbFilter.setNumberOfIntegrations(lRow.getNumberOfIntegrations());
        lJaxbFilter.setReadoutPattern(NirCamReadPattern_JaxbReadoutPatternType.getValueFromKey(lRow.getReadoutPattern()));
        lJaxbFilter.setRequestedExposureTime(lRow.getRequestedExpTime());

        lJaxbFilters.getFilterConfig().add(lJaxbFilter);
    }
    lJaxbTemplate.setFilters(lJaxbFilters);

    return lJaxbTemplate;
}

public static void convertToDm(JaxbNircamCoron iJaxbTemplate, NirCamCoronTemplate oTemplate) {
    oTemplate.setAcqTargetFromString(iJaxbTemplate.getAcqTargetID());
    oTemplate.setAcqFilter(NirCamFilter_JaxbFilterType.getKeyFromValue(iJaxbTemplate.getAcqFilter()));
    oTemplate.setMask(NirCamMask_JaxbNircamMask.getKeyFromValue(iJaxbTemplate.getCoronMask()));

    for (edu.stsci.jwst.apt.template.nircamcoron.JaxbFilterConfigType lJaxbFilter :
        iJaxbTemplate.getFilters().getFilterConfig()) {
        NirCamCoronExposureSpecTableRow lRow = new NirCamCoronExposureSpecTableRow();
        oTemplate.addExposure(lRow);

        lRow.setFilter(NirCamFilter_JaxbFilterType.getKeyFromValue(lJaxbFilter.getFilter()));
        lRow.setNumberOfGroups(lJaxbFilter.getNumberOfGroups());
        lRow.setNumberOfIntegrations(lJaxbFilter.getNumberOfIntegrations());
        lRow.setReadoutPattern(NirCamReadPattern_JaxbReadoutPatternType.getKeyFromValue(lJaxbFilter.getReadoutPattern()));
        lRow.setRequestedExpTime(lJaxbFilter.getRequestedExposureTime());
    }
}

public static JaxbNircamInternalFlat convertToJaxb(NirCamFlatTemplate iTemplate) {
    JaxbNircamInternalFlat lJaxbTemplate = new JaxbNircamInternalFlat();

    lJaxbTemplate.setModule(NirCamModule_JaxbModuleType.getValueFromKey(iTemplate.getModule()));

    edu.stsci.jwst.apt.template.nircaminternalflat.JaxbFiltersType lJaxbFilters =
```

```
        new edu.stsci.jwst.apt.template.nircaminternalflat.JaxbFiltersType();

    for(NirCamFlatExposureSpecTableRow lRow : iTemplate.getExposures()) {
        edu.stsci.jwst.apt.template.nircaminternalflat.JaxbFilterConfigType lJaxbFilter
            = new edu.stsci.jwst.apt.template.nircaminternalflat.JaxbFilterConfigType();

        lJaxbFilter.setLongFilter(NirCamFilter_JaxbFilterType.getValueFromKey(lRow.getLongFilter()));
        lJaxbFilter.setShortFilter(NirCamFilter_JaxbFilterType.getValueFromKey(lRow.getShortFilter()));
        lJaxbFilter.setIntegrations(lRow.getNumberOfIntegrations());
        lJaxbFilter.setExposures(lRow.getNumberOfExposures());
        lJaxbFilter.setActualExposureTime(lRow.getActualExpTime());

        lJaxbFilters.getFilterConfig().add(lJaxbFilter);
    }
    lJaxbTemplate.setFilters(lJaxbFilters);

    return lJaxbTemplate;
}

public static void convertToDm(JaxbNircamInternalFlat iJaxbTemplate, NirCamFlatTemplate oTemplate) {
    oTemplate.setModule(NirCamModule_JaxbModuleType.getKeyFromValue(iJaxbTemplate.getModule()));

    for (edu.stsci.jwst.apt.template.nircaminternalflat.JaxbFilterConfigType lJaxbFilter :
        iJaxbTemplate.getFilters().getFilterConfig()) {
        NirCamFlatExposureSpecTableRow lRow = new NirCamFlatExposureSpecTableRow();
        oTemplate.addExposure(lRow);

        lRow.setLongFilter(NirCamFilter_JaxbFilterType.getKeyFromValue(lJaxbFilter.getLongFilter()));
        lRow.setShortFilter(NirCamFilter_JaxbFilterType.getKeyFromValue(lJaxbFilter.getShortFilter()));
        lRow.setNumberOfIntegrations(lJaxbFilter.getIntegrations());
        lRow.setNumberOfExposures(lJaxbFilter.getExposures());
    }
}

public static JaxbNircamWheelExercise convertToJaxb(NirCamWheelExerciseTemplate iTemplate) {
    JaxbNircamWheelExercise lJaxbTemplate = new JaxbNircamWheelExercise();

    lJaxbTemplate.setMechanism(NirCamMechanism_JaxbMechanismType.getValueFromKey(iTemplate.getMechType()));
    lJaxbTemplate.setRotations(iTemplate.getNumRotations());

    for (NirCamWheel lWheel : iTemplate.getWheels()) {
        lJaxbTemplate.getWheel().add(NirCamWheel_JaxbWheelType.getValueFromKey(lWheel));
    }

    return lJaxbTemplate;
}

public static void convertToDm(JaxbNircamWheelExercise iJaxbTemplate, NirCamWheelExerciseTemplate oTemplate) {
    oTemplate.setMechType(NirCamMechanism_JaxbMechanismType.getKeyFromValue(iJaxbTemplate.getMechanism()));
    oTemplate.setNumRotations(iJaxbTemplate.getRotations());
```

```
        Set<NirCamWheel> lWheels = new TreeSet<NirCamWheel>();
        for (JaxbWheelType lWheel : iJaxbTemplate.getWheel()) {
            lWheels.add(NirCamWheel_JaxbWheelType.getKeyFromValue(lWheel));
        }
        oTemplate.setWheels(lWheels);
    }

    public static JaxbNircamDark convertToJaxb(NirCamDarkTemplate iTemplate) {
        JaxbNircamDark lJaxbTemplate = new JaxbNircamDark();

        lJaxbTemplate.setModule(NirCamModule_JaxbModuleType.getValueFromKey(iTemplate.getModule()));
        lJaxbTemplate.setReadoutRegion(NirCamSubarray_JaxbReadoutRegionType.getValueFromKey(iTemplate.getSubarray()));

        JaxbExposuresType lJaxbExposures = new JaxbExposuresType();

        for (NirCamDarkExposureSpecTableRow lRow : iTemplate.getExposures()) {
            JaxbExposureConfigType lJaxbExposure = new JaxbExposureConfigType();

            lJaxbExposure.setExposures(lRow.getNumberOfExposures());
            lJaxbExposure.setGroups(lRow.getNumberOfGroups());
            lJaxbExposure.setIntegrations(lRow.getNumberOfIntegrations());
            lJaxbExposure.setReadoutPattern(NirCamReadPattern_JaxbReadoutPatternType.getValueFromKey(lRow.getReadoutPattern()));

            lJaxbExposures.getExposureConfig().add(lJaxbExposure);
        }
        lJaxbTemplate.setExposures(lJaxbExposures);

        return lJaxbTemplate;
    }

    public static void convertToDm(JaxbNircamDark iJaxbTemplate, NirCamDarkTemplate oTemplate) {
        oTemplate.setModule(NirCamModule_JaxbModuleType.getKeyFromValue(iJaxbTemplate.getModule()));
        oTemplate.setSubarray(NirCamSubarray_JaxbReadoutRegionType.getKeyFromValue(iJaxbTemplate.getReadoutRegion()));

        for (JaxbExposureConfigType lJaxbExposure : iJaxbTemplate.getExposures().getExposureConfig()) {
            NirCamDarkExposureSpecTableRow lRow = new NirCamDarkExposureSpecTableRow();
            oTemplate.addExposure(lRow);

            lRow.setNumberOfExposures(lJaxbExposure.getExposures());
            lRow.setNumberOfGroups(lJaxbExposure.getGroups());
            lRow.setNumberOfIntegrations(lJaxbExposure.getIntegrations());
            lRow.setReadoutPattern(NirCamReadPattern_JaxbReadoutPatternType.getKeyFromValue(lJaxbExposure.getReadoutPattern()));
        }
    }

    public static JaxbNircamFocus convertToJaxb(NirCamFocusTemplate iTemplate) {
        JaxbNircamFocus lJaxbTemplate = new JaxbNircamFocus();

        lJaxbTemplate.setModule(NirCamModule_JaxbModuleType.getValueFromKey(iTemplate.getModule()));
```

```
        lJaxbTemplate.setShortFilter(NirCamFilter_JaxbFilterType.getValueFromKey(iTemplate.getShortFilter()));
        lJaxbTemplate.setLongFilter(NirCamFilter_JaxbFilterType.getValueFromKey(iTemplate.getLongFilter()));
        lJaxbTemplate.setShortPupil(NirCamPupil_JaxbPupilType.getValueFromKey(iTemplate.getShortPupil()));
        lJaxbTemplate.setLongPupil(NirCamPupil_JaxbPupilType.getValueFromKey(iTemplate.getLongPupil()));

        lJaxbTemplate.setGroups(iTemplate.getNumberOfGroups());
        lJaxbTemplate.setIntegrations(iTemplate.getNumberOfIntegrations());
        lJaxbTemplate.setReadoutPattern(NirCamReadPattern_JaxbReadoutPatternType.getValueFromKey(iTemplate.getReadoutPattern()));

        JaxbLinearActuatorType lSteps = new JaxbLinearActuatorType();
        lSteps.setValue1(iTemplate.getStartSteps(1));
        lSteps.setValue2(iTemplate.getStartSteps(2));
        lSteps.setValue3(iTemplate.getStartSteps(3));
        lJaxbTemplate.setStartingPositionSteps(lSteps);

        JaxbLinearActuatorType lUnits = new JaxbLinearActuatorType();
        lUnits.setValue1(iTemplate.getStartSensorUnits(1));
        lUnits.setValue2(iTemplate.getStartSensorUnits(2));
        lUnits.setValue3(iTemplate.getStartSensorUnits(3));
        lJaxbTemplate.setStartingPositionSensorUnits(lUnits);

        JaxbLinearActuatorType lMotorPhase = new JaxbLinearActuatorType();
        lMotorPhase.setValue1(iTemplate.getStartMotorPhase(1));
        lMotorPhase.setValue2(iTemplate.getStartMotorPhase(2));
        lMotorPhase.setValue3(iTemplate.getStartMotorPhase(3));
        lJaxbTemplate.setStartingMotorPhase(lMotorPhase);

        lJaxbTemplate.setPositions(new JaxbPositionsType());
        for (NirCamLinearActuatorTableRow lRow : iTemplate.getLAPositions()) {
            JaxbLinearActuatorType lActuator = new JaxbLinearActuatorType();
            lActuator.setValue1(lRow.getLAPosition(1));
            lActuator.setValue2(lRow.getLAPosition(2));
            lActuator.setValue3(lRow.getLAPosition(3));
            lJaxbTemplate.getPositions().getPosition().add(lActuator);
        }

        lJaxbTemplate.setReturnToStart(iTemplate.getReturnToStart());

        return lJaxbTemplate;
    }

    public static void convertToDm(JaxbNircamFocus iJaxbTemplate, NirCamFocusTemplate oTemplate) {
        oTemplate.setModule(NirCamModule_JaxbModuleType.getKeyFromValue(iJaxbTemplate.getModule()));

        oTemplate.setShortFilter(NirCamFilter_JaxbFilterType.getKeyFromValue(iJaxbTemplate.getShortFilter()));
        oTemplate.setLongFilter(NirCamFilter_JaxbFilterType.getKeyFromValue(iJaxbTemplate.getLongFilter()));
        oTemplate.setShortPupil(NirCamPupil_JaxbPupilType.getKeyFromValue(iJaxbTemplate.getShortPupil()));
        oTemplate.setLongPupil(NirCamPupil_JaxbPupilType.getKeyFromValue(iJaxbTemplate.getLongPupil()));
```

```
    oTemplate.setNumberOfGroups(iJaxbTemplate.getGroups());
    oTemplate.setNumberOfIntegrations(iJaxbTemplate.getIntegrations());
    oTemplate.setReadoutPattern(NirCamReadPattern_JaxbReadoutPatternType.getKeyFromValue(iJaxbTemplate.getReadoutPattern()));

    if (iJaxbTemplate.getStartingPositionSteps() != null) {
        oTemplate.setStartSteps(1, iJaxbTemplate.getStartingPositionSteps().getValue1());
        oTemplate.setStartSteps(2, iJaxbTemplate.getStartingPositionSteps().getValue2());
        oTemplate.setStartSteps(3, iJaxbTemplate.getStartingPositionSteps().getValue3());
    }

    if (iJaxbTemplate.getStartingPositionSensorUnits() != null) {
        oTemplate.setStartSensorUnits(1, iJaxbTemplate.getStartingPositionSensorUnits().getValue1());
        oTemplate.setStartSensorUnits(2, iJaxbTemplate.getStartingPositionSensorUnits().getValue2());
        oTemplate.setStartSensorUnits(3, iJaxbTemplate.getStartingPositionSensorUnits().getValue3());
    }

    if (iJaxbTemplate.getStartingMotorPhase() != null) {
        oTemplate.setStartMotorPhase(1, iJaxbTemplate.getStartingMotorPhase().getValue1());
        oTemplate.setStartMotorPhase(2, iJaxbTemplate.getStartingMotorPhase().getValue2());
        oTemplate.setStartMotorPhase(3, iJaxbTemplate.getStartingMotorPhase().getValue3());
    }

    if (iJaxbTemplate.getPositions() != null) {
        for (JaxbLinearActuatorType lActuator : iJaxbTemplate.getPositions().getPosition()) {
            NirCamLinearActuatorTableRow lRow = new NirCamLinearActuatorTableRow();
            oTemplate.addLAPosition(lRow);

            lRow.setLAPosition(1, lActuator.getValue1());
            lRow.setLAPosition(2, lActuator.getValue2());
            lRow.setLAPosition(3, lActuator.getValue3());
        }
    }

    oTemplate.setReturnToStart(iJaxbTemplate.isReturnToStart());
}

public static JaxbNircamWheelThresholdCurrent convertToJaxb(NirCamWheelThresholdCurrentTemplate iTemplate) {
    JaxbNircamWheelThresholdCurrent lJaxbTemplate = new JaxbNircamWheelThresholdCurrent();

    return lJaxbTemplate;
}

public static void convertToDm(JaxbNircamWheelThresholdCurrent iJaxbTemplate, NirCamWheelThresholdCurrentTemplate oTemplate) {

}
//[end]
//[end]

//[start]--------------------JAXB <---> DM Mapping--------
```

```
// This class is used internally to maintain a two-way map between JAXB enums and DM enums.  It
// is assumed the values entered into this map maintain a one-to-one relationship.
private static class TwoWayHashMap<T, U> {

    public void put(T key, U value) {
        initialMap.put(key, value);
        reverseMap.put(value, key);
    }

    public U getValueFromKey(T key) {
        return initialMap.get(key);
    }

    public T getKeyFromValue(U key) {
        return reverseMap.get(key);
    }

    public Set<T> getKeys() {
        return initialMap.keySet();
    }

    private HashMap<T, U> initialMap = new HashMap<T, U>();
    private HashMap<U, T> reverseMap = new HashMap<U, T>();
}

//   [start] Miri Mappings
private static final TwoWayHashMap<MiriObjectType, JaxbObjectTypeType> MiriObjectType_JaxbObjectTypeType =
    new TwoWayHashMap<MiriObjectType, JaxbObjectTypeType>();
static {
    MiriObjectType_JaxbObjectTypeType.put(MiriObjectType.BRIGHT, JaxbObjectTypeType.BRIGHT);
    MiriObjectType_JaxbObjectTypeType.put(MiriObjectType.FAINT, JaxbObjectTypeType.FAINT);
}

private static final TwoWayHashMap<MiriSubarray, JaxbSubarrayType> MiriSubarray_JaxbSubarrayType =
    new TwoWayHashMap<MiriSubarray, JaxbSubarrayType>();
static {
    MiriSubarray_JaxbSubarrayType.put(MiriSubarray.BRIGHTSKY, JaxbSubarrayType.BRIGHTSKY);
    MiriSubarray_JaxbSubarrayType.put(MiriSubarray.FULL, JaxbSubarrayType.FULL);
}

private static final TwoWayHashMap<MiriFilter, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType> MiriFilter_JaxbFilterType =
    new TwoWayHashMap<MiriFilter, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType>();
static {
    MiriFilter_JaxbFilterType.put(MiriFilter.F560W, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_560_W);
    MiriFilter_JaxbFilterType.put(MiriFilter.F770W, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_770_W);
    MiriFilter_JaxbFilterType.put(MiriFilter.F1000W, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1000_W);
    MiriFilter_JaxbFilterType.put(MiriFilter.F1065C, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1065_C);
    MiriFilter_JaxbFilterType.put(MiriFilter.F1130W, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1130_W);
    MiriFilter_JaxbFilterType.put(MiriFilter.F1140C, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1140_C);
    MiriFilter_JaxbFilterType.put(MiriFilter.F1280W, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1280_W);
```

```
        MiriFilter_JaxbFilterType.put(MiriFilter.F1500W, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1500_W);
        MiriFilter_JaxbFilterType.put(MiriFilter.F1550C, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1550_C);
        MiriFilter_JaxbFilterType.put(MiriFilter.F1800W, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1800_W);
        MiriFilter_JaxbFilterType.put(MiriFilter.F2100W, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_2100_W);
        MiriFilter_JaxbFilterType.put(MiriFilter.F2300C, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_2300_C);
        MiriFilter_JaxbFilterType.put(MiriFilter.F2550W, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_2550_W);
        MiriFilter_JaxbFilterType.put(MiriFilter.F2550WR, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_2550_WR);
        MiriFilter_JaxbFilterType.put(MiriFilter.P750L, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.P_750_L);
        MiriFilter_JaxbFilterType.put(MiriFilter.FND, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.FND);
    }

    private static final TwoWayHashMap<MiriCoronFilter, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType>
MiriFilter_JaxbCoronFilterType =
        new TwoWayHashMap<MiriCoronFilter, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType>();
    static {
        MiriFilter_JaxbCoronFilterType.put(MiriCoronFilter._4QPMF1065C, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1065_C);
        MiriFilter_JaxbCoronFilterType.put(MiriCoronFilter._4QPMF1140C, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1140_C);
        MiriFilter_JaxbCoronFilterType.put(MiriCoronFilter._4QPMF1550C, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_1550_C);
        MiriFilter_JaxbCoronFilterType.put(MiriCoronFilter.LYOTF2300C, edu.stsci.jwst.apt.instrument.miri.JaxbFilterType.F_2300_C);
    }

    private static final TwoWayHashMap<MiriReadPattern, JaxbReadoutPatternType> MiriReadPattern_JaxbReadoutPatternType =
        new TwoWayHashMap<MiriReadPattern, JaxbReadoutPatternType>();
    static {
        MiriReadPattern_JaxbReadoutPatternType.put(MiriReadPattern.FAST, JaxbReadoutPatternType.FAST);
        MiriReadPattern_JaxbReadoutPatternType.put(MiriReadPattern.SLOW, JaxbReadoutPatternType.SLOW);
    }

    private static final TwoWayHashMap<MiriFlatSuite, JaxbFlatSuiteType> MiriFlatSuite_JaxbFlatSuiteType =
        new TwoWayHashMap<MiriFlatSuite, JaxbFlatSuiteType>();
    static {
        MiriFlatSuite_JaxbFlatSuiteType.put(MiriFlatSuite.ALL, JaxbFlatSuiteType.ALL);
        MiriFlatSuite_JaxbFlatSuiteType.put(MiriFlatSuite.ONE, JaxbFlatSuiteType.ONE);
        MiriFlatSuite_JaxbFlatSuiteType.put(MiriFlatSuite.PRINCIPAL, JaxbFlatSuiteType.PRINCIPAL);
    }

    private static final TwoWayHashMap<MiriWavelength, JaxbWavelengthType> MiriWavelength_JaxbWavelengthType =
        new TwoWayHashMap<MiriWavelength, JaxbWavelengthType>();
    static {
        MiriWavelength_JaxbWavelengthType.put(MiriWavelength.LONG, JaxbWavelengthType.LONG);
        MiriWavelength_JaxbWavelengthType.put(MiriWavelength.MEDIUM, JaxbWavelengthType.MEDIUM);
        MiriWavelength_JaxbWavelengthType.put(MiriWavelength.SHORT, JaxbWavelengthType.SHORT);
        MiriWavelength_JaxbWavelengthType.put(MiriWavelength.ALL, JaxbWavelengthType.ALL);
    }

    private static final TwoWayHashMap<MiriDetector, JaxbDetectorType> MiriDetector_JaxbDetectorType =
        new TwoWayHashMap<MiriDetector, JaxbDetectorType>();
    static {
        MiriDetector_JaxbDetectorType.put(MiriDetector.ALL, JaxbDetectorType.ALL);
        MiriDetector_JaxbDetectorType.put(MiriDetector.IMAGER, JaxbDetectorType.IMAGER);
```

```
    MiriDetector_JaxbDetectorType.put(MiriDetector.MRS, JaxbDetectorType.MRS);
}

private static final TwoWayHashMap<MiriMask, JaxbMaskType> MiriMask_JaxbMaskType =
    new TwoWayHashMap<MiriMask, JaxbMaskType>();
static {
    MiriMask_JaxbMaskType.put(MiriMask._4QPM, JaxbMaskType.FOUR_QPM);
    MiriMask_JaxbMaskType.put(MiriMask.LYOT, JaxbMaskType.LYOT);
}
//   [end]

//   [start] Nircam Mappings
private static final TwoWayHashMap<NirCamModule, JaxbModuleType> NirCamModule_JaxbModuleType =
    new TwoWayHashMap<NirCamModule, JaxbModuleType>();
static {
    NirCamModule_JaxbModuleType.put(NirCamModule.A, JaxbModuleType.A);
    NirCamModule_JaxbModuleType.put(NirCamModule.B, JaxbModuleType.B);
    NirCamModule_JaxbModuleType.put(NirCamModule.ALL, JaxbModuleType.ALL);
}

private static final TwoWayHashMap<NirCamSubarray, JaxbReadoutRegionType> NirCamSubarray_JaxbReadoutRegionType =
    new TwoWayHashMap<NirCamSubarray, JaxbReadoutRegionType>();
static {
    NirCamSubarray_JaxbReadoutRegionType.put(NirCamSubarray.FULL, JaxbReadoutRegionType.FULL);
    NirCamSubarray_JaxbReadoutRegionType.put(NirCamSubarray.SUB16, JaxbReadoutRegionType.SUB_16);
    NirCamSubarray_JaxbReadoutRegionType.put(NirCamSubarray.SUB320, JaxbReadoutRegionType.SUB_320);
    NirCamSubarray_JaxbReadoutRegionType.put(NirCamSubarray.SUB48, JaxbReadoutRegionType.SUB_48);
    NirCamSubarray_JaxbReadoutRegionType.put(NirCamSubarray.SUB640, JaxbReadoutRegionType.SUB_640);
    NirCamSubarray_JaxbReadoutRegionType.put(NirCamSubarray.SUB96, JaxbReadoutRegionType.SUB_96);
}

private static final TwoWayHashMap<NirCamFilter, JaxbFilterType> NirCamFilter_JaxbFilterType =
    new TwoWayHashMap<NirCamFilter, JaxbFilterType>();
static {
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F277W, JaxbFilterType.F_277_W);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F322W2, JaxbFilterType.F_322_W_2);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F356W, JaxbFilterType.F_356_W);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F444W, JaxbFilterType.F_444_W);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F250M, JaxbFilterType.F_250_M);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F300M, JaxbFilterType.F_300_M);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F335M, JaxbFilterType.F_335_M);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F360M, JaxbFilterType.F_360_M);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F410M, JaxbFilterType.F_410_M);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F430M, JaxbFilterType.F_410_M);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F460M, JaxbFilterType.F_460_M);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F480M, JaxbFilterType.F_480_M);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F323N_F356W, JaxbFilterType.F_323_N_F_356_W);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F323N_F322W2, JaxbFilterType.F_323_N_F_322_W_2);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F405N_F444W, JaxbFilterType.F_405_N_F_444_W);
    NirCamFilter_JaxbFilterType.put(NirCamFilter.F405N_F410M, JaxbFilterType.F_405_N_F_410_M);
```

```
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F418N_F444W, JaxbFilterType.F_418_N_F_444_W);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F418N_F410M, JaxbFilterType.F_418_N_F_410_M);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F466N_F444W, JaxbFilterType.F_466_N_F_444_W);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F466N_F460M, JaxbFilterType.F_466_N_F_460_M);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F470N_F444W, JaxbFilterType.F_470_N_F_444_W);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F070W, JaxbFilterType.F_070_W);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F090W, JaxbFilterType.F_090_W);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F115W, JaxbFilterType.F_115_W);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F150W, JaxbFilterType.F_150_W );
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F150W2, JaxbFilterType.F_150_W_2);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F200W, JaxbFilterType.F_200_W);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F140M, JaxbFilterType.F_140_M);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F182M, JaxbFilterType.F_182_M);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F210M, JaxbFilterType.F_210_M);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F187N, JaxbFilterType.F_187_N);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F212N, JaxbFilterType.F_212_N);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.WL3, JaxbFilterType.WL_3);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F162M_F150W2, JaxbFilterType.F_162_M_F_150_W_2);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F164N_F150W, JaxbFilterType.F_164_N_F_150_W);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F164N_F150W2, JaxbFilterType.F_164_N_F_150_W_2);
        NirCamFilter_JaxbFilterType.put(NirCamFilter.F225N_F150W2, JaxbFilterType.F_225_N_F_150_W_2);
    }

    private static final TwoWayHashMap<NirCamPupil, JaxbPupilType> NirCamPupil_JaxbPupilType =
        new TwoWayHashMap<NirCamPupil, JaxbPupilType>();
    static {
        NirCamPupil_JaxbPupilType.put(NirCamPupil.CLEAR, JaxbPupilType.CLEAR);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.FLAT, JaxbPupilType.FLAT);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.CORON1, JaxbPupilType.CORON_1);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.CORON2, JaxbPupilType.CORON_2);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.PINHOLES, JaxbPupilType.PINHOLES);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.F164N, JaxbPupilType.F_164_N);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.F162M, JaxbPupilType.F_162_M);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.F225N, JaxbPupilType.F_225_N);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.GDHS1, JaxbPupilType.GDHS_1);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.GDHS2, JaxbPupilType.GDHS_2);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.WL1, JaxbPupilType.WL_1);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.WL2, JaxbPupilType.WL_2);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.F323N, JaxbPupilType.F_323_N);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.F418N, JaxbPupilType.F_418_N);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.F470N, JaxbPupilType.F_470_N);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.F405N, JaxbPupilType.F_405_N);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.F466N, JaxbPupilType.F_466_N);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.GRISM1, JaxbPupilType.GRISM_1);
        NirCamPupil_JaxbPupilType.put(NirCamPupil.GRISM2, JaxbPupilType.GRISM_2);
    }

    private static final TwoWayHashMap<NirCamReadPattern, edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType>
NirCamReadPattern_JaxbReadoutPatternType =
        new TwoWayHashMap<NirCamReadPattern, edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType>();
```

```
    static {
        NirCamReadPattern_JaxbReadoutPatternType.put(NirCamReadPattern.DEEP8,
edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType.DEEP_8);
        NirCamReadPattern_JaxbReadoutPatternType.put(NirCamReadPattern.DEEP2,
edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType.DEEP_2);
        NirCamReadPattern_JaxbReadoutPatternType.put(NirCamReadPattern.MEDIUM8,
edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType.MEDIUM_8);
        NirCamReadPattern_JaxbReadoutPatternType.put(NirCamReadPattern.MEDIUM2,
edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType.MEDIUM_2);
        NirCamReadPattern_JaxbReadoutPatternType.put(NirCamReadPattern.SHALLOW4,
edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType.SHALLOW_4);
        NirCamReadPattern_JaxbReadoutPatternType.put(NirCamReadPattern.SHALLOW2,
edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType.SHALLOW_2);
        NirCamReadPattern_JaxbReadoutPatternType.put(NirCamReadPattern.BRIGHT2,
edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType.BRIGHT_2);
        NirCamReadPattern_JaxbReadoutPatternType.put(NirCamReadPattern.BRIGHT1,
edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType.BRIGHT_1);
        NirCamReadPattern_JaxbReadoutPatternType.put(NirCamReadPattern.RAPID,
edu.stsci.jwst.apt.instrument.nircam.JaxbReadoutPatternType.RAPID);
    }

    private static final TwoWayHashMap<NirCamMechanism, JaxbMechanismType> NirCamMechanism_JaxbMechanismType =
        new TwoWayHashMap<NirCamMechanism, JaxbMechanismType>();
    static {
        NirCamMechanism_JaxbMechanismType.put(NirCamMechanism.FILTER, JaxbMechanismType.FILTER);
        NirCamMechanism_JaxbMechanismType.put(NirCamMechanism.PUPIL, JaxbMechanismType.PUPIL);
    }

    private static final TwoWayHashMap<NirCamWheel, JaxbWheelType> NirCamWheel_JaxbWheelType =
        new TwoWayHashMap<NirCamWheel, JaxbWheelType>();
    static {
        NirCamWheel_JaxbWheelType.put(NirCamWheel.ALL, JaxbWheelType.ALL);
        NirCamWheel_JaxbWheelType.put(NirCamWheel.LONGA, JaxbWheelType.LONGA);
        NirCamWheel_JaxbWheelType.put(NirCamWheel.LONGB, JaxbWheelType.LONGB);
        NirCamWheel_JaxbWheelType.put(NirCamWheel.SHORTA, JaxbWheelType.SHORTA);
        NirCamWheel_JaxbWheelType.put(NirCamWheel.SHORTB, JaxbWheelType.SHORTB);
    }

    private static final TwoWayHashMap<NirCamMask, JaxbCoronMaskType> NirCamMask_JaxbNircamMask =
        new TwoWayHashMap<NirCamMask, JaxbCoronMaskType>();
    static {
        NirCamMask_JaxbNircamMask.put(NirCamMask.MASK210R, JaxbCoronMaskType.MASK_210_R);
        NirCamMask_JaxbNircamMask.put(NirCamMask.MASK335R, JaxbCoronMaskType.MASK_335_R);
        NirCamMask_JaxbNircamMask.put(NirCamMask.MASK430R, JaxbCoronMaskType.MASK_430_R);
        NirCamMask_JaxbNircamMask.put(NirCamMask.MASKLWB, JaxbCoronMaskType.MASKLWB);
        NirCamMask_JaxbNircamMask.put(NirCamMask.MASKSWB, JaxbCoronMaskType.MASKSWB);
    }
    //   [end]
    //[end]
}
```

```java
package edu.stsci.jwst.apt.io;

import gov.nasa.gsfc.util.MessageLogger;


/**
 * JwstProposalFileConverter
 *
 * This class provides a mechanism for upgrading a .APT file for a JWST Proposal.
 *
 */
public class JwstProposalFileConverter {

    // Don't ever instantiate this class
    private JwstProposalFileConverter() {}

    /**
     * Takes an APT file and converts it to the specified version.  Down-converting is not supported
     * @param iFile The APT file to convert
     * @param iToVersion The version to which this file should be converted
     * @return 1) The original File if iToVersion is the same as the file's version<br>
     *         2) A new file when conversion is successful<br>
     *         3) null if iToVersion is less than the file's current version, or if there's a problem
     */
    public static Document convertFile(File iFile, int iToVersion) {
        try {

            Document lDoc = getDocumentFromFile(iFile);

            int lOrigVersion = getVersion(lDoc);
            if(lOrigVersion == iToVersion) {
                return lDoc;
            }

            // We don't downgrade
            if(lOrigVersion > iToVersion) {
                return null;
            }

            MessageLogger.getInstance().writeDebug(null, "Upgrading JWST Proposal from version ["+getVersion(lDoc)+"] to version
["+iToVersion+"]");
            Document lNewDoc = convertDoc(lDoc, iToVersion);

            return lNewDoc;

        }
        catch(Exception ex) {
            ex.printStackTrace();
            return null;
        }
```

```
}

private static Document getDocumentFromFile(File iFile) {
    try {
        DocumentBuilderFactory lFactory = DocumentBuilderFactory.newInstance();
        lFactory.setNamespaceAware(true);
        DocumentBuilder lDocBuilder = lFactory.newDocumentBuilder();

        return lDocBuilder.parse(iFile);
    }
    catch (Exception ex) {
        ex.printStackTrace();
        return null;
    }
}

private static int getVersion(Document iDoc) {
    return Integer.parseInt(iDoc.getDocumentElement().getAttribute("schemaVersion"));
}

private static Document convertDoc(Document iDoc, int iToVersion) {

    // This case should never happen, but check it anyway to guard against infinite recursion
    if(getVersion(iDoc) > iToVersion) {
        throw new InvalidParameterException(
                "The document's version ["+getVersion(iDoc)+"] is greater than the schema version ["+iToVersion+"]");
    }

    if(getVersion(iDoc) == iToVersion) {
        return iDoc;
    } else {
        return convertDoc(upgradeDoc(iDoc), iToVersion);
    }
}

private static Document upgradeDoc(Document iDoc) {
    switch(getVersion(iDoc)) {
        case 1:
            return convertToVersion2(iDoc);
        case 2:
            return convertToVersion3(iDoc);
        case 3:
            iDoc.getDocumentElement().setAttribute("schemaVersion", "4");
            return iDoc;

        default:
            return null;
    }
}
```

```
    private static Document convertToVersion3(Document iDocV2) {
        // Version 2 used the following format:
        //      <ns11:AcqFilter>
        //        <ns12:Filter>F335M</ns12:Filter>
        //      </ns11:AcqFilter>
        // where ns11 is the Nircam Coron schema, ns12 is the Nircam Instrument schema
        //
        // We want the schema to become this:
        //      <ns11:AcqFilter>F335M</ns11:AcqFilter>
        //
        NodeList oldNircamCoronAcqTargets = iDocV2.getElementsByTagNameNS("http://www.stsci.edu/JWST/APT/Template/NircamCoron",
"AcqFilter");
        for (int i=0; i<oldNircamCoronAcqTargets.getLength(); i++) {
            Element targetAcqElement = (Element)oldNircamCoronAcqTargets.item(i);
            String filter = getElementValue(targetAcqElement, "Filter", "http://www.stsci.edu/JWST/APT/Instrument/Nircam");

            // Remove the ns12 Filter node
            Node child = targetAcqElement.getElementsByTagNameNS("*", "Filter").item(0);
            if (child != null) {
                targetAcqElement.removeChild(child);
                targetAcqElement.getFirstChild().setNodeValue(filter);

                // There is a second #text node that's hanging around, and I'm not sure why.  This gets
                // rid of it so there's only one #text node: the filter we just set.
                targetAcqElement.removeChild(targetAcqElement.getChildNodes().item(1));
            }
            // If we somehow have an old AcqFilter element with no Filter child, delete the AcqFilter
            // element.
            else {
                targetAcqElement.getParentNode().removeChild(targetAcqElement);
            }
        }

        //
        // Remove "AcqFlux" elements
        //
        NodeList acqFluxElements = iDocV2.getElementsByTagNameNS("*", "AcqFlux");
        while (acqFluxElements.getLength() > 0) {
            Element acqFlux = (Element)acqFluxElements.item(0);
            acqFlux.getParentNode().removeChild(acqFlux);
        }

        acqFluxElements = iDocV2.getElementsByTagNameNS("*", "AcqFilterFlux");
        while (acqFluxElements.getLength() > 0) {
            Element acqFlux = (Element)acqFluxElements.item(0);
            acqFlux.getParentNode().removeChild(acqFlux);
        }

        //
        // Remove the "Filter" element from the MIRI Dark templates
```

```
        //
        NodeList miriDarkFilters = iDocV2.getElementsByTagNameNS("http://www.stsci.edu/JWST/APT/Template/MiriDark", "Filter");
        while (miriDarkFilters.getLength() > 0) {
            Element filter = (Element)miriDarkFilters.item(0);
            filter.getParentNode().removeChild(filter);
        }

        //
        // We're fickle on Target Names.  Get rid of the (Fixed Target: Equatorial) part
        //
//        NodeList obsNodes = iDocV2.getElementsByTagName("Observation");
//        for (int i=0; i<obsNodes.getLength(); i++) {
//            Element obsElement = (Element)obsNodes.item(i);
//            String targID = getElementValue(obsElement, "TargetID");
//            if (targID != null) {
//                setElementValue(obsElement, "TargetID", targID.substring(0, targID.indexOf(" (Fixed Target: Equatorial)")));
//            }
//        }

        iDocV2.getDocumentElement().setAttribute("schemaVersion", "3");
        return iDocV2;
    }

    private static Document convertToVersion2(Document iDocV1) {
//        debugPrintElement(iDocV1, "");

        //
        // Conversion: Change the "Imager" tag names to "Detector" for Miri Anneal templates
        //
        NodeList miriImagerNodes = iDocV1.getElementsByTagNameNS("http://www.stsci.edu/JWST/APT/Template/MiriAnneal", "Imager");
        for (int i=0; i<miriImagerNodes.getLength(); i++) {
            Node miriImagerNode = miriImagerNodes.item(i);
            iDocV1.renameNode(miriImagerNode, "http://www.stsci.edu/JWST/APT/Template/MiriAnneal", "Detector");
        }

        //
        // Conversion: Change Target names to match the proper format: <targ num> <targ name> (Fixed Target: Equatorial)"
        //     In Version 1, all targets are fixed, equatorial.  Also, they only specified the <targ name>, so they would
        //     save properly, but they would not load.  Also, Acq Targets were not in the picture yet.
        //
        HashMap<String, String> targNameMap = new HashMap<String, String>();
        NodeList targetNodes = iDocV1.getElementsByTagName("Target");
        for (int i=0; i<targetNodes.getLength(); i++) {
            Element targetElement = (Element)targetNodes.item(i);

            String targNum = getElementValue(targetElement, "Number");
            String targName = getElementValue(targetElement, "TargetName");

            if (targNum != null && targNum.length() > 0 &&
                    targName != null && targName.length() > 0) {
```

```
            targNameMap.put(targName, targNum + " " + targName + " (Fixed Target: Equatorial)");
        }
    }

    NodeList obsNodes = iDocV1.getElementsByTagName("Observation");
    for (int i=0; i<obsNodes.getLength(); i++) {
        Element obsElement = (Element)obsNodes.item(i);
        String targID = getElementValue(obsElement, "TargetID");
        if (targID != null) {
            setElementValue(obsElement, "TargetID", targNameMap.get(targID));
        }
    }


    iDocV1.getDocumentElement().setAttribute("schemaVersion", "2");
    return iDocV1;
}

// Muddle through all the DOM complexity to get what we really want.  Element is a proper
// element, iTagName is the unique tag of a child element.  In the following example,
// value is returned:
//
//  <iElement>
//     <iTagName>value</iTagName>
//  </iElement>
private static String getElementValue (Element iElement, String iTagName) {
    return getElementValue (iElement, iTagName, "*");
}

private static String getElementValue (Element iElement, String iTagName, String iChildNamespace) {
    if (iElement == null ||
        iElement.getElementsByTagNameNS(iChildNamespace, iTagName) == null ||
        iElement.getElementsByTagNameNS(iChildNamespace, iTagName).item(0) == null ||
        iElement.getElementsByTagNameNS(iChildNamespace, iTagName).item(0).getFirstChild() == null) {
        return null;
    }

    return iElement.getElementsByTagNameNS(iChildNamespace, iTagName).item(0).getFirstChild().getNodeValue();
}

// Muddle through all the DOM complexity to get what we really want.  Element is a proper
// element, iTagName is the unique tag of a child element, and iValue is the value to set.
// In the following example, the element with tag iTagName's value is set to iValue
//
//  <iElement>
//     <iTagName>iValue</iTagName>
//  </iElement>
private static void setElementValue (Element iElement, String iTagName, String iValue) {
    iElement.getElementsByTagName(iTagName).item(0).getFirstChild().setNodeValue(iValue);
}
```

```
    // This is a developer debug message that prints the hierarchy of the document
    private static void debugPrintElement(Node iNode, String iIndent) {
        StringBuilder lAttribs = new StringBuilder();
        if(iNode == null) {
            return;
        }

        if(iNode.getAttributes() != null) {
            for(int i=0; i<iNode.getAttributes().getLength(); i++) {
                Node attrib = iNode.getAttributes().item(i);
                if(i > 0) {
                    lAttribs.append(", ");
                }
                lAttribs.append(attrib.getNodeName());
                lAttribs.append(":");
                lAttribs.append(attrib.getNodeValue());
            }
        }

        iNode.getAttributes();
        System.out.println(iIndent + iNode.getNodeName() + "[" + lAttribs.toString() + "]: " + iNode.getNodeValue());

        for(int i=0; i<iNode.getChildNodes().getLength(); i++) {
            Node n = iNode.getChildNodes().item(i);
            debugPrintElement(n, iIndent + "  ");
        }
    }
}
```

```java
package edu.stsci.jwst.apt.io;

import java.io.*;

public class SqlExporter {

    //[start]--------------------Statics----------------------
    private static final String PROG_ID = "program_id";
    private static final String OBS_NUM = "observation_number";
    private static final String VISIT_ID = "visit_id";
    private static final String TEMPLATE = "template";

    private static final String READOUT_PATTERN = "readout_pattern";
    private static final String MASK = "mask";
    private static final String NUM_INTS = "number_of_integrations";
    private static final String DETECTOR = "detector";
    private static final String NUM_GRPS = "number_of_groups";
    private static final String FLAT_SUITE = "flat_suite";
    private static final String WAVELEN = "wavelength";
    private static final String SUBARRAY = "subarray";
    private static final String NUM_EXP = "number_of_exposures";

    private static final String FILTER = "filter";
    private static final String NUM_INT = "number_of_integrations";
    //[end]

    //[start]--------------------Fields-----------------------
    private enum EDatabaseTable {
        PROGRAM,
        PROGRAM_TEXT,
        OBSERVATION,
        TARGET,
        TARGET_FLUX,
        FIXED_TARGET,
        VISIT,
        VISIT_TARGETS,
        PHASE2_EXPOSURE,
        MIRI_TEMPLATES,
        MIRI_SPECTRAL_ELEMENTS,
        MIRI_TARGET_ACQ,
        NIRCAM_TEMPLATES,
        NIRCAM_FILTERS,
        NIRCAM_TARGET_ACQ
    }

    // Developer's Note:
    //
    // Due to the possibly complex relation between database tables, a design decision was made to
    // simplify the code by requiring the developer to determine the order in which rows from tables
    // should be deleted.  This Vector specifies the order in which rows in the tables should be
```

```
// deleted, with the first element being the first table from which to delete.
private Vector<EDatabaseTable> fTablesSortedForDeletion = new Vector<EDatabaseTable>();
{
    fTablesSortedForDeletion.add(EDatabaseTable.NIRCAM_FILTERS);
    fTablesSortedForDeletion.add(EDatabaseTable.NIRCAM_TARGET_ACQ);
    fTablesSortedForDeletion.add(EDatabaseTable.NIRCAM_TEMPLATES);
    fTablesSortedForDeletion.add(EDatabaseTable.MIRI_SPECTRAL_ELEMENTS);
    fTablesSortedForDeletion.add(EDatabaseTable.MIRI_TARGET_ACQ);
    fTablesSortedForDeletion.add(EDatabaseTable.MIRI_TEMPLATES);
    fTablesSortedForDeletion.add(EDatabaseTable.PHASE2_EXPOSURE);
    fTablesSortedForDeletion.add(EDatabaseTable.VISIT_TARGETS);
    fTablesSortedForDeletion.add(EDatabaseTable.VISIT);
    fTablesSortedForDeletion.add(EDatabaseTable.FIXED_TARGET);
    fTablesSortedForDeletion.add(EDatabaseTable.TARGET_FLUX);
    fTablesSortedForDeletion.add(EDatabaseTable.TARGET);
    fTablesSortedForDeletion.add(EDatabaseTable.OBSERVATION);
    fTablesSortedForDeletion.add(EDatabaseTable.PROGRAM_TEXT);
    fTablesSortedForDeletion.add(EDatabaseTable.PROGRAM);
}

// This is a list of template names as they should appear in the database
private HashMap<Class<?>, String> templateNameLookup = new HashMap<Class<?>, String>();
{
    templateNameLookup.put(MiriImagingTemplate.class, "MIRI Imaging");
    templateNameLookup.put(MiriAnnealTemplate.class, "MIRI Anneal");
    templateNameLookup.put(MiriDarkTemplate.class, "MIRI Dark");
    templateNameLookup.put(MiriImagingFlatTemplate.class, "MIRI Imaging Flat");
    templateNameLookup.put(MiriMrsFlatTemplate.class, "MIRI MRS Flat");
    templateNameLookup.put(MiriCoronTemplate.class, "MIRI Coronagraphic Imaging");
    templateNameLookup.put(MiriLrsTemplate.class, "MIRI Low Resolutin Spectroscopy");
    templateNameLookup.put(MiriMrsTemplate.class, "MIRI Medium Resolution Spectroscopy");

    templateNameLookup.put(NirCamImagingTemplate.class, "NIRCam Imaging");
    templateNameLookup.put(NirCamCoronTemplate.class, "NIRCam Coronagraphic Imaging");
    templateNameLookup.put(NirCamDarkTemplate.class, "NIRCam Dark");
    templateNameLookup.put(NirCamFlatTemplate.class, "NIRCam Internal Flat");
    templateNameLookup.put(NirCamWheelExerciseTemplate.class, "NIRCam Wheel Exercise");
    templateNameLookup.put(NirCamFocusTemplate.class, "NIRCam Focus");
    templateNameLookup.put(NirCamWheelThresholdCurrentTemplate.class, "NIRCam Wheel Threshold Current");
}

private File fFile = null;
private JwstProposalSpecification fProp = null;

// Records to be inserted into the database.  Note that the order elements are
// inserted into this Vector are the order they will be inserted into the database.
private Vector<AbstractDatabaseRecord> fRecords = new Vector<AbstractDatabaseRecord>();
//[end]

//[start]--------------------Constructors-----------------
```

```
public SqlExporter(File iOutputFile, JwstProposalSpecification iProp) {
    fFile = iOutputFile;
    fProp = iProp;
}
//[end]

//[start]--------------------Accessors--------------------
private int getProgramID() {
    return fProp.getPropInfo().getProposalID();
}

private List<JwstObservation> getObservations() {
    Vector<JwstObservation> lObservations = new Vector<JwstObservation>();
    for (JwstObservationGroup lObsGroup : fProp.getDataRequestFolder().getChildren(JwstObservationGroup.class)) {
        for (JwstObservation lObs : lObsGroup.getChildren(JwstObservation.class)) {
            lObservations.add(lObs);
        }
    }
    return lObservations;
}

private List<JwstVisit> getVisits() {
    Vector<JwstVisit> lVisits = new Vector<JwstVisit>();
    for (JwstObservation lObs : getObservations()) {
        for (JwstVisit lVisit : lObs.getChildren(JwstVisit.class)) {
            lVisits.add(lVisit);
        }
    }
    return lVisits;
}
//[end]

//[start]--------------------Methods----------------------
/**
 * This method exports the JwstProposalSpecification to the File specified in the constructor.
 * An exception is thrown if the proposal was not successfully written to the file.
 * @throws IOException, InvalidParameterException
 */
public void export() throws IOException {
    PrintWriter fOut = null;
    try {
        if (fFile == null) {
            throw new InvalidParameterException("Invalid Configuration: File cannot be null.");
        }
        if (fProp == null) {
            throw new InvalidParameterException("Invalid Configuration: Proposal cannot be null.");
        }
        if (fProp.getPropInfo() == null) {
            throw new InvalidParameterException("Invalid Configuration: Proposal Information cannot be null.");
        }
```

```
        if (fProp.getPropInfo().getProposalID() == null) {
            throw new InvalidParameterException("Invalid Configuration: Proposal ID cannot be null.");
        }

        // Create database records from the proposal
        // Note: These tables should be in the order they are to be inserted into the database
        processProposalTable();
        processObservationTable();
        processTargetTables();
        processVisitTable();
        processVisitTargetsTable();

        processTemplates();

        fOut = new PrintWriter(new BufferedWriter(new FileWriter(fFile)));

        // Generate the SQL
        fOut.println("declare @Error int");
        fOut.println("set @Error = 0");
        fOut.println("begin transaction");
        fOut.println();

        // Delete old records
        for (EDatabaseTable lTable : fTablesSortedForDeletion) {
            fOut.println("delete from " + lTable.name().toLowerCase() + " where " + PROG_ID + " = " + getProgramID());
            fOut.println("set @Error = @Error | @@error");
            fOut.println();
        }

        // Insert new records
        for (AbstractDatabaseRecord lRecord : fRecords) {
            fOut.println(lRecord.createInsertSqlStatement());
            fOut.println("set @Error = @Error | @@error");
            fOut.println();
        }

        fOut.println("if @Error = 0   commit");
        fOut.println("else            rollback");
        fOut.println();
    }
    finally {
        if (fOut != null) {
            fOut.close();
        }
    }
}

private void processProposalTable() {
    ProgramRecord lRecord = new ProgramRecord(fProp);
    ProgramTextRecord lTextRecord = new ProgramTextRecord(fProp);
```

```
        fRecords.add(lRecord);
        fRecords.add(lTextRecord);
    }

    private void processObservationTable() {
        for (JwstObservation lObs : getObservations()) {
            ObservationRecord lRecord = new ObservationRecord(lObs);
            fRecords.add(lRecord);
        }
    }

    private void processTargetTables() {
        for (JwstFixedTarget lTarg : fProp.getTargets().getChildren(JwstFixedTarget.class)) {
            TargetRecord lRecord = new TargetRecord(lTarg);
            fRecords.add(lRecord);

            int index = 1;
            for (AcqFluxTableEntry lFlux : lTarg.getAcqFluxes()) {
                TargetFluxRecord lFluxRecord = new TargetFluxRecord(lTarg, lFlux, index++);
                fRecords.add(lFluxRecord);
            }

            FixedTargetRecord lFixedRecord = new FixedTargetRecord(lTarg);
            fRecords.add(lFixedRecord);
        }
    }

    private void processVisitTable() {
        for (JwstVisit lVisit : getVisits()) {
            VisitRecord lRecord = new VisitRecord(lVisit);
            fRecords.add(lRecord);
        }
    }

    private void processVisitTargetsTable() {
        for (JwstVisit lVisit : getVisits()) {
            // TODO: The "getTarget" should really be called on the Visit, not the Observation
            if (lVisit.getObservation().getTarget() != null) {
                VisitTargetsRecord lRecord = new VisitTargetsRecord(lVisit, ((NumberedTarget)lVisit.getObservation().getTarget
()).getNumber());
                fRecords.add(lRecord);
            }
        }
    }

    // This method creates a Template row for each visit.  It also populates Spectral Element rows based on
    // the template type and the Phase 2 Exposure Table (requested/actual exposure times).
    private void processTemplates() {
```

```
for (JwstVisit lVisit : getVisits()) {

    if (lVisit.getTemplate() != null) {

        //
        // MIRI Templates
        //
        if (lVisit.getTemplate().getInstrument() == MiriInstrument.getInstance()) {
            MiriTemplatesRecord lTemplateRecord = new MiriTemplatesRecord(lVisit);

            // Note: We need to add the record at this point for two reasons.  First, it must appear in the
            // SQL script before we add the corresponding Spectral Element rows.  Second, even if there are no
            // specific fields defined (as is the case for Miri Coron, LRS, MRS), we still need a row in the
            // MIRI_TEMPLATES table for the rows in MIRI_SPECTRAL_ELEMENTS table to reference.
            //
            // Note that while we are adding the record now, we don't finish populating it until later in
            // this method.
            fRecords.add(lTemplateRecord);

            JwstTemplate lTemplate = lVisit.getTemplate();

            // Add template-specific fields
            if (lTemplate instanceof MiriAnnealTemplate) {
                MiriAnnealTemplate lAnnealTemplate = (MiriAnnealTemplate)lTemplate;
                lTemplateRecord.put(DETECTOR, lAnnealTemplate.getDetector());
            } else if (lTemplate instanceof MiriDarkTemplate) {

                MiriDarkTemplate lDarkTemplate = (MiriDarkTemplate)lTemplate;

                lTemplateRecord.put(DETECTOR, lDarkTemplate.getDetector());

                int index = 1;
                for (MiriDarkExposureSpecTableRow lExp : lDarkTemplate.getExposures()) {
                    MiriSpectralElementsRecord lSpecEltRecord = new MiriSpectralElementsRecord(lVisit, lExp, index++);
                    lSpecEltRecord.put(NUM_EXP, lExp.getNumberOfExposures());

                    fRecords.add(lSpecEltRecord);
                    fRecords.add(new Phase2ExposureRecord(lVisit.getObservation().getNumber(), lVisit.getNumber(), index - 1,
                            lExp.getRequestedExpTime(), lExp.getActualExpTime()));
                }
            } else if (lTemplate instanceof MiriImagingFlatTemplate) {
                MiriImagingFlatTemplate lFlatTemplate = (MiriImagingFlatTemplate) lTemplate;

                lTemplateRecord.put(FLAT_SUITE, lFlatTemplate.getFlatSuite());

                MiriSpectralElementsRecord lSpecEltRecord = new MiriSpectralElementsRecord(lVisit, 1);

                lSpecEltRecord.put(FILTER, lFlatTemplate.getFilter());
                lSpecEltRecord.put(NUM_INT, lFlatTemplate.getNumInts());
```

```
            fRecords.add(lSpecEltRecord);
        } else if (lTemplate instanceof MiriImagingTemplate) {
            MiriImagingTemplate lMiriTemplate = (MiriImagingTemplate) lTemplate;

            lTemplateRecord.put(SUBARRAY, lMiriTemplate.getSubarray());

            int index = 1;
            for (MiriImagingExposureSpecTableRow lExp : lMiriTemplate.getExposures()) {
                MiriSpectralElementsRecord lSpecEltRecord = new MiriSpectralElementsRecord(lVisit, lExp, index++);
                fRecords.add(lSpecEltRecord);
                fRecords.add(new Phase2ExposureRecord(lVisit.getObservation().getNumber(), lVisit.getNumber(), index - 1,
                        lExp.getRequestedExpTime(), lExp.getActualExpTime()));
            }
        } else if (lTemplate instanceof MiriMrsFlatTemplate) {
            MiriMrsFlatTemplate lFlatTemplate = (MiriMrsFlatTemplate)lTemplate;
            lTemplateRecord.put(WAVELEN, lFlatTemplate.getWavelength());
            lTemplateRecord.put(FLAT_SUITE, lFlatTemplate.getFlatSuite());

            MiriSpectralElementsRecord lSpecEltRecord = new MiriSpectralElementsRecord(lVisit, 1);
            lSpecEltRecord.put(NUM_INT, lFlatTemplate.getNumberOfIntegrations());

            fRecords.add(lSpecEltRecord);
        } else if (lTemplate instanceof MiriCoronTemplate) {
            // TODO: This is still in the works.  Need to figure out what to populate at this point
        } else if (lTemplate instanceof MiriLrsTemplate) {
            MiriSpectralElementsRecord lSpecEltRecord = new MiriSpectralElementsRecord(lVisit, 1);
            MiriLrsTemplate lLrsTemplate = (MiriLrsTemplate) lTemplate;

            lSpecEltRecord.put(NUM_INT, lLrsTemplate.getNumberOfIntegrations());
            lSpecEltRecord.put(READOUT_PATTERN, lLrsTemplate.getReadoutPattern());
            lSpecEltRecord.put(NUM_GRPS, lLrsTemplate.getNumberOfGroups());

            fRecords.add(lSpecEltRecord);
            fRecords.add(new Phase2ExposureRecord(lVisit.getObservation().getNumber(), lVisit.getNumber(), 1,
                    lLrsTemplate.getRequestedExpTime(), lLrsTemplate.getActualExpTime()));

        } else if (lTemplate instanceof MiriMrsTemplate) {
            MiriMrsTemplate lMrsTemplate = (MiriMrsTemplate) lTemplate;

            lTemplateRecord.put(WAVELEN, lMrsTemplate.getWavelength());

            MiriSpectralElementsRecord lSpecEltRecord = new MiriSpectralElementsRecord(lVisit, 1);
            lSpecEltRecord.put(NUM_INT, lMrsTemplate.getNumberOfIntegrations());
            lSpecEltRecord.put(READOUT_PATTERN, lMrsTemplate.getReadoutPattern());
            lSpecEltRecord.put(NUM_GRPS, lMrsTemplate.getNumberOfGroups());

            fRecords.add(lSpecEltRecord);
            fRecords.add(new Phase2ExposureRecord(lVisit.getObservation().getNumber(), lVisit.getNumber(), 1,
                    lMrsTemplate.getRequestedExpTime(), lMrsTemplate.getActualExpTime()));
        }
```

```
    // Add Target Acq record if this is a Target Acq template.  Note this is orthogonal to
    // the specific instance of the templates handled above
    if (lTemplate instanceof MiriTargetAcqTemplate) {
        MiriTargetAcqTemplate lAcqTemplate = (MiriTargetAcqTemplate)lTemplate;
        MiriTargetAcqRecord lRecord = new MiriTargetAcqRecord(lVisit, lAcqTemplate);

        fRecords.add(lRecord);
    }
}


//
// Nircam Templates
//
else if (lVisit.getTemplate().getInstrument() == NirCamInstrument.getInstance()) {
    NircamTemplatesRecord lTemplateRecord = new NircamTemplatesRecord(lVisit);
    fRecords.add(lTemplateRecord);

    JwstTemplate lTemplate = lVisit.getTemplate();

    if (lTemplate instanceof NirCamImagingTemplate) {
        NirCamImagingTemplate lNircamTemplate = (NirCamImagingTemplate) lTemplate;

        lTemplateRecord.put("readout_region", lNircamTemplate.getSubarray());
        lTemplateRecord.put("modules", lNircamTemplate.getModule());

        int index = 1;
        for (NirCamImagingExposureSpecTableRow lExp : lNircamTemplate.getExposures()) {
            NircamFiltersRecord lFiltersRecord = new NircamFiltersRecord(lVisit, lExp, index++);
            fRecords.add(lFiltersRecord);
            fRecords.add(new Phase2ExposureRecord(lVisit.getObservation().getNumber(), lVisit.getNumber(), index - 1,
                    lExp.getRequestedExpTime(), lExp.getActualExpTime()));
        }
    }
    else if (lTemplate instanceof NirCamCoronTemplate) {
        NirCamCoronTemplate lNircamTemplate = (NirCamCoronTemplate) lTemplate;

        lTemplateRecord.put("occulting_mask", lNircamTemplate.getMask());

        int index = 1;
        for (NirCamCoronExposureSpecTableRow lExp : lNircamTemplate.getExposures()) {
            NircamFiltersRecord lFiltersRecord = new NircamFiltersRecord(lVisit, lExp, index++);

            lFiltersRecord.put("filter", lExp.getFilter());
            fRecords.add(lFiltersRecord);
            fRecords.add(new Phase2ExposureRecord(lVisit.getObservation().getNumber(), lVisit.getNumber(), index - 1,
                    lExp.getRequestedExpTime(), lExp.getActualExpTime()));
        }
    }
```

```
else if (lTemplate instanceof NirCamDarkTemplate) {
    NirCamDarkTemplate lNircamTemplate = (NirCamDarkTemplate) lTemplate;

    lTemplateRecord.put("modules", lNircamTemplate.getModule());
    lTemplateRecord.put("readout_region", lNircamTemplate.getSubarray());

    int index = 1;
    for (NirCamDarkExposureSpecTableRow lExp : lNircamTemplate.getExposures()) {
        NircamFiltersRecord lFiltersRecord = new NircamFiltersRecord(lVisit, lExp, index++);
        lFiltersRecord.put(NUM_EXP, lExp.getNumberOfExposures());
        fRecords.add(lFiltersRecord);
        fRecords.add(new Phase2ExposureRecord(lVisit.getObservation().getNumber(), lVisit.getNumber(), index - 1,
                lExp.getRequestedExpTime(), lExp.getActualExpTime()));
    }
}
else if (lTemplate instanceof NirCamFlatTemplate) {
    NirCamFlatTemplate lNircamTemplate = (NirCamFlatTemplate) lTemplate;

    lTemplateRecord.put("modules", lNircamTemplate.getModule());

    int index = 1;
    for (NirCamFlatExposureSpecTableRow lExp : lNircamTemplate.getExposures()) {
        NircamFiltersRecord lFiltersRecord = new NircamFiltersRecord(lVisit, lExp, index++);
        lFiltersRecord.put(NUM_EXP, lExp.getNumberOfExposures());
        fRecords.add(lFiltersRecord);
        fRecords.add(new Phase2ExposureRecord(lVisit.getObservation().getNumber(), lVisit.getNumber(), index - 1,
                lExp.getRequestedExpTime(), lExp.getActualExpTime()));
    }
}
else if (lTemplate instanceof NirCamFocusTemplate) {
    NirCamFocusTemplate lNircamTemplate = (NirCamFocusTemplate) lTemplate;

    lTemplateRecord.put("modules", lNircamTemplate.getModule());

    NircamFiltersRecord lFiltersRecord = new NircamFiltersRecord(lVisit, 1);
    lFiltersRecord.put("filter_short_A", lNircamTemplate.getShortFilter());
    lFiltersRecord.put("filter_long_A", lNircamTemplate.getLongFilter());
    lFiltersRecord.put("pupil_short_A", lNircamTemplate.getShortPupil());
    lFiltersRecord.put("pupil_long_A", lNircamTemplate.getLongPupil());
    lFiltersRecord.put("readout_pattern", lNircamTemplate.getReadoutPattern());
    lFiltersRecord.put("number_of_groups", lNircamTemplate.getNumberOfGroups());
    lFiltersRecord.put("number_of_integrations", lNircamTemplate.getNumberOfIntegrations());
    lTemplateRecord.put("return_to_start", lNircamTemplate.getReturnToStart());

    fRecords.add(lFiltersRecord);
    fRecords.add(new Phase2ExposureRecord(lVisit.getObservation().getNumber(), lVisit.getNumber(), 1,
            0.0, lNircamTemplate.getActualExpTime()));
}
else if (lTemplate instanceof NirCamWheelExerciseTemplate) {
    NirCamWheelExerciseTemplate lNircamTemplate = (NirCamWheelExerciseTemplate) lTemplate;
```

```
                    lTemplateRecord.put("wheel_type", lNircamTemplate.getMechType());
                    lTemplateRecord.put("number_of_rotations", lNircamTemplate.getNumRotations());

                    // TODO: Need to populate specific wheels.  Right now, the list goes into a single db field,
                    // and that's just lame.
                }

                if (lTemplate instanceof NirCamTargetAcqTemplate) {
                    NirCamTargetAcqTemplate lAcqTemplate = (NirCamTargetAcqTemplate)lTemplate;
                    NircamTargetAcqRecord lRecord = new NircamTargetAcqRecord(lVisit, lAcqTemplate);

                    fRecords.add(lRecord);
                }

            }
            //[end] Nircam Templates
        }
    }
}
//[end]

//[start]--------------------Inner Classes----------------
private abstract class AbstractDatabaseRecord {
    private LinkedHashMap<String, String> fFieldMap = new LinkedHashMap<String, String>();

    public abstract EDatabaseTable getTableName();

    public void put(String iColumn, Enum iValue) {
        if (iValue != null) {
            put(iColumn, iValue.name());
        } else {
            put(iColumn, "");
        }
    }

    public void put(String iColumn, String iValue) {
        fFieldMap.put(iColumn, convertToSqlString(iValue));
    }

    public void put(String iColumn, Integer iValue) {
        fFieldMap.put(iColumn, convertToSqlString(iValue));
    }

    public void put(String iColumn, Double iValue) {
        fFieldMap.put(iColumn, convertToSqlString(iValue));
    }
    // The SQL database is naively using character values "Y" and "N" when it should really
    // be using Boolean values.  As such, we're forced to accommodate that behavior here.
    public void put(String iColumn, Boolean iValue) {
```

```
    if (iValue != null) {
        put(iColumn, iValue == true ? "Y" : "N");
    } else {
        put(iColumn, "");
    }
}

public String get(String iColumn) {
    return fFieldMap.get(iColumn);
}

public String remove(String iColumn) {
    return fFieldMap.remove(iColumn);
}

public String createInsertSqlStatement() {
    StringBuffer lBuffer = new StringBuffer();

    lBuffer.append("insert into ");
    lBuffer.append(getTableName().name().toLowerCase());
    lBuffer.append(" ( ");

    // Insert column names
    Iterator<String> lColIter = fFieldMap.keySet().iterator();
    while (lColIter.hasNext()) {
        String lColName = lColIter.next();
        lBuffer.append(lColName);
        if (lColIter.hasNext()) {
            lBuffer.append(", ");
        } else {
            lBuffer.append(" ) ");
        }
    }

    lBuffer.append("values ( ");

    // Insert values
    lColIter = fFieldMap.keySet().iterator();
    while (lColIter.hasNext()) {
        String lColName = lColIter.next();
        String lValue = fFieldMap.get(lColName);

        lBuffer.append(lValue);
        if (lColIter.hasNext()) {
            lBuffer.append(", ");
        } else {
            lBuffer.append(" )");
        }
    }
```

```
            return lBuffer.toString();
        }

        // We need to export a String to the SQL file.  In order to do so, certain conversions
        // may need to occur.  For example, Strings need to be wrapped in single quotes.
        // All the necessary conversions between Java objects and SQL Strings occur in these
        // overloaded "convertToSqlString" methods.
        private String convertToSqlString (String iString) {
            if (iString == null) {
                iString = "";
            }
            return "'" + iString.replace("'", "''") + "'";
        }

        private String convertToSqlString (Integer iInteger) {
            if (iInteger == null) {
                return convertToSqlString("");
            }
            else {
                return Integer.toString(iInteger);
            }
        }

        private String convertToSqlString (Double iDouble) {
            if (iDouble == null) {
                return convertToSqlString("");
            }
            else {
                return Double.toString(iDouble);
            }
        }
    }

    private class MiriSpectralElementsRecord extends AbstractDatabaseRecord {

        public MiriSpectralElementsRecord(JwstVisit iVisit, int recordId) {
            this(iVisit, null, recordId);
        }

        public MiriSpectralElementsRecord(JwstVisit iVisit, MiriExposureSpecTableRow iExp, int recordId) {
            put(PROG_ID, getProgramID());
            put(OBS_NUM, iVisit.getObservation().getNumber());
            put(VISIT_ID, iVisit.getNumber());
            put("spectral_element_record_id", recordId);

            if (iExp != null) {
                put(FILTER, iExp.getFilter());
                put(READOUT_PATTERN, iExp.getReadoutPattern());
                put(NUM_GRPS, iExp.getNumberOfGroups());
                put(NUM_INTS, iExp.getNumberOfIntegrations());
```

```
        }
        else {
            put(FILTER, "");
            put(READOUT_PATTERN, "");
            put(NUM_GRPS, "");
            put(NUM_INTS, "");
        }
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.MIRI_SPECTRAL_ELEMENTS;
    }
}

private class MiriTargetAcqRecord extends AbstractDatabaseRecord {
    public MiriTargetAcqRecord(JwstVisit iVisit, MiriTargetAcqTemplate iTemplate) {
        put(PROG_ID, getProgramID());
        put(OBS_NUM, iVisit.getObservation().getNumber());
        put(VISIT_ID, iVisit.getNumber());
        put("target_acq_record_id", 1);
        if (iTemplate.getAcqTarget() != null) {
            put("target_id", ((NumberedTarget)iTemplate.getAcqTarget()).getNumber());

            int fluxId = 0;
            List<AcqFluxTableEntry> lFluxList = ((JwstFixedTarget)iTemplate.getAcqTarget()).getAcqFluxes();
            for (int i=0; i<lFluxList.size(); i++) {
                AcqFluxTableEntry lFlux = lFluxList.get(i);
                if (lFlux.getFilter() == iTemplate.getAcqFilter())
                {
                    fluxId = i+1;
                    break;
                }
            }

            if (fluxId > 0) {
                put("flux_id", fluxId);
            }

            put("filter", iTemplate.getAcqFilter().name());
        }
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.MIRI_TARGET_ACQ;
    }
}

private class Phase2ExposureRecord extends AbstractDatabaseRecord {
```

```
    public Phase2ExposureRecord(Integer obsNum, Integer visitID, Integer expId, Double reqExpTime, Double actExpTime) {
        put(PROG_ID, getProgramID());
        put("observation_number", obsNum);
        put("visit_id", visitID);
        put("exposure_record_id", expId);
        put("requested_exposure_duration", reqExpTime);
        put("actual_exposure_time", actExpTime);
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.PHASE2_EXPOSURE;
    }
}

private abstract class TemplatesRecord extends AbstractDatabaseRecord {
    public TemplatesRecord(JwstVisit iVisit) {
        put(PROG_ID, getProgramID());
        put(OBS_NUM, iVisit.getObservation().getNumber());
        put(VISIT_ID, iVisit.getNumber());
    }
}

private class MiriTemplatesRecord extends TemplatesRecord {

    public MiriTemplatesRecord(JwstVisit iVisit) {
        super(iVisit);
        addCommonFields();
    }

    // Initialization code - since the database is completely incapable of handing NULL values, we have to
    // pre-populate all the fields with empty values.  This is a convenience for the developer.
    private void addCommonFields() {
        put(MASK, "");
        put(DETECTOR, "");
        put(FLAT_SUITE, "");
        put(WAVELEN, "");
        put(SUBARRAY, "");
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.MIRI_TEMPLATES;
    }
}

private class NircamTemplatesRecord extends TemplatesRecord {
    public NircamTemplatesRecord(JwstVisit iVisit) {
        super(iVisit);
        addCommonFields();
```

```
    }

    private void addCommonFields() {
        put("bulb", "");
        put("readout_region", "");
        put("occulting_mask", "");
        put("detectors", "");
        put("calibration_assembly", "");
        put("current_level", "");
        put("line_actuator_starting_steps", "");
        put("line_actuator_starting_sensor_units", "");
        put("starting_motor_phases", "");
        put("return_to_start", "Y");
        put("wheel_type", "");
        put("modules", "");
        put("number_of_rotations", "");
        put("channel", "");
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.NIRCAM_TEMPLATES;
    }
}

private class NircamFiltersRecord extends AbstractDatabaseRecord {

    public NircamFiltersRecord(JwstVisit iVisit, int recordId) {
        this(iVisit, null, recordId);
    }
    public NircamFiltersRecord(JwstVisit iVisit, NirCamExposureSpecTableRow iExp, int recordId) {
        put(PROG_ID, getProgramID());
        put(OBS_NUM, iVisit.getObservation().getNumber());
        put(VISIT_ID, iVisit.getNumber());

        put("filter_short_B", "");
        put("filter_long_B", "");
        put("pupil_long_A", "");
        put("pupil_long_B", "");
        put("pupil_short_A", "");
        put("pupil_short_B", "");
        put("filter", "");
        put("filter_record_id", recordId);

        if (iExp != null) {
            put(READOUT_PATTERN, iExp.getReadoutPattern());
            put("number_of_groups", iExp.getNumberOfGroups());
            put("number_of_integrations", iExp.getNumberOfIntegrations());
            put("filter_short_A", iExp.getShortFilter());
            put("filter_long_A", iExp.getLongFilter());
```

```
        }
        else {
            put(READOUT_PATTERN, "");
            put("number_of_groups", "");
            put("number_of_integrations", "");
            put("filter_short_A", "");
            put("filter_long_A", "");
        }
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.NIRCAM_FILTERS;
    }
}

private class NircamTargetAcqRecord extends AbstractDatabaseRecord {
    public NircamTargetAcqRecord(JwstVisit iVisit, NirCamTargetAcqTemplate iTemplate) {
        put(PROG_ID, getProgramID());
        put(OBS_NUM, iVisit.getObservation().getNumber());
        put(VISIT_ID, iVisit.getNumber());

        put("target_acq_readout_pattern", "");
        put("target_acq_number_of_integrations", "");
        put("long_visit_id", "");
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.NIRCAM_TARGET_ACQ;
    }
}

private class VisitRecord extends AbstractDatabaseRecord {
    public VisitRecord(JwstVisit iVisit) {
        JwstObservation lObs = (JwstObservation) iVisit.getParent();
        put(PROG_ID, getProgramID());
        put(OBS_NUM, lObs.getNumber());
        put(VISIT_ID, iVisit.getNumber());
        put(TEMPLATE, templateNameLookup.get(iVisit.getTemplate().getClass()));

        put("internal_target", "N");
        put("external_target", "Y");
        put("comment", "<VISIT COMMENT - COMPLETE THIS SECTION>");
        put("on_hold_comment", "<ON HOLD COMMENT - COMPLETE THIS SECTION>");

    }

    @Override
    public EDatabaseTable getTableName() {
```

```
        return EDatabaseTable.VISIT;
    }
}


private class ObservationRecord extends AbstractDatabaseRecord {
    public ObservationRecord(JwstObservation iObservation) {
        put(PROG_ID, getProgramID());
        put(OBS_NUM, iObservation.getNumber());
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.OBSERVATION;
    }
}

private class TargetRecord extends AbstractDatabaseRecord {
    public TargetRecord (JwstFixedTarget iTarget) {
        put(PROG_ID, getProgramID());
        put("target_id", iTarget.getNumber());
        put("target_name", iTarget.getName());
        put("standard_target_name", "");
        put("target_type", "F");
        put("target_description", "");
        put("comment", "");
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.TARGET;
    }
}

private class TargetFluxRecord extends AbstractDatabaseRecord {
    public TargetFluxRecord (JwstFixedTarget iTarget, AcqFluxTableEntry iFlux, int iFluxID) {
        put(PROG_ID, getProgramID());
        put("target_id", iTarget.getNumber());
        put("flux_id", iFluxID);
        if (iFlux.getInstrument() != null) {
            put("flux_type", iFlux.getInstrument().getName() + "/" + iFlux.getFilterAsString());
        } else {
            put("flux_type", "");
        }
        put("flux_value", iFlux.getFlux());
        put("flux_uncertainty", "");
        put("flux_units", "microJy");
    }

    @Override
```

```
    public EDatabaseTable getTableName() {
        return EDatabaseTable.TARGET_FLUX;
    }
}

private class FixedTargetRecord extends AbstractDatabaseRecord {
    public FixedTargetRecord (JwstFixedTarget iTarget) {
        put(PROG_ID, getProgramID());
        put("target_id", iTarget.getNumber());
        put("position_reference", "ABSOLUTE");
        put("parent_target_id", 0);
        put("ra_computed", iTarget.getCoordinates().getRa());
        put("dec_computed", iTarget.getCoordinates().getDec());
        put("ra_uncertainty_computed", iTarget.getRaUncDegrees());
        put("dec_uncertainty_computed", iTarget.getDecUncDegrees());
        put("ra_proper_motion", "");
        put("dec_proper_motion", "");
        put("epoch", "");
        put("parallax", "");
        put("parallax_uncertainty", "");
        put("ra_literal", "");
        put("dec_literal", "");
        put("ra_uncertainty_literal", "");
        put("dec_uncertainty_literal", "");
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.FIXED_TARGET;
    }
}

private class VisitTargetsRecord extends AbstractDatabaseRecord {
    public VisitTargetsRecord (JwstVisit iVisit, int iTargID) {
        put(PROG_ID, getProgramID());
        put("observation_number", iVisit.getObservation().getNumber());
        put("visit_id", iVisit.getNumber());
        put("target_id", iTargID);
    }

    @Override
    public EDatabaseTable getTableName() {
        return EDatabaseTable.VISIT_TARGETS;
    }
}

private class ProgramTextRecord extends AbstractDatabaseRecord {
    public ProgramTextRecord(JwstProposalSpecification iProp) {
        put(PROG_ID, getProgramID());
        //TODO: Sanitation needed?
```

```
            put("abstract", iProp.getPropInfo().getAbstract());
        }

        @Override
        public EDatabaseTable getTableName() {
            return EDatabaseTable.PROGRAM_TEXT;
        }
    }

    private class ProgramRecord extends AbstractDatabaseRecord {
        public ProgramRecord(JwstProposalSpecification iProp) {
            put(PROG_ID, getProgramID());
            put("title", iProp.getPropInfo().getTitle());
            put("program_type", iProp.getPropInfo().getCategory());
            put("program_subtype", iProp.getPropInfo().getSubCategory());
        }

        @Override
        public EDatabaseTable getTableName() {
            return EDatabaseTable.PROGRAM;
        }
    }
    //[end]
}
```