A database for TMT interface control documents

Kim Gillies*, Scott Roberts, Allan Brighton, John Rogers
Thirty Meter Telescope International Observatory, 100 West Walnut St., Suite 300, Pasadena CA, 91124, USA

ABSTRACT

The TMT Software System consists of software components that interact with one another through a software infrastructure called TMT Common Software (CSW). CSW consists of software services and library code that is used by developers to create the subsystems and components that participate in the software system. CSW also defines the types of components that can be constructed and their roles. The use of common component types and shared middleware services allows standardized software interfaces for the components.

A software system called the TMT Interface Database System was constructed to support the documentation of the interfaces for components based on CSW. The programmer describes a subsystem and each of its components using JSON-style text files. A command interface file describes each command a component can receive and any commands a component sends. The event interface files describe status, alarms, and events a component publishes and status and events subscribed to by a component. A web application was created to provide a user interface for the required features. Files are ingested into the software system's database. The user interface allows browsing subsystem interfaces, publishing versions of subsystem interfaces, and constructing and publishing interface control documents that consist of the intersection of two subsystem interfaces. All published subsystem interfaces and interface control documents are versioned for configuration control and follow the standard TMT change control processes. Subsystem interfaces and interface control documents can be visualized in the browser or exported as PDF files.

Keywords: software interface, database, ICD

1. INTRODUCTION

From a software communications and integration viewpoint, the TMT Software System consists of a set of software components interacting with each other through a software communications backbone and software infrastructure (middleware). The integration of all these software components requires software infrastructure that is outside the scope of the individual components. The design of the TMT Software System has resulted in a small set of services and associated software called TMT Common Software that is focused on the task of integrating software components.

The individual software components have a wide range of purposes within the software system. Some will be standalone, software-only applications; others will be software components within complex software/hardware subsystems. The CSW defines a set of standard components with specified roles that reduce the amount of documentation needed to describe a design and the component interfaces.

An expected arrangement of software components is shown in Figure 1. At the highest level are sequencers that send commands to Assemblies. An Assembly represents user level controllable hardware. An example is a filter wheel in an instrument. Assemblies control the low-level hardware devices through one or more hardware control daemons or HCDs. Each HCD encapsulates the specific protocols needed to communicate with some hardware, which is usually networked.

Two CSW services support communication between components. The Connection and Command Service (CCS) supports sending and receiving commands between two components. In the TMT software design, an Application or Sequence Component connects to Assemblies and commands them by submitting configurations. Assemblies connect to and command their associated HCDs. Commands are sent with a peer to peer connection between two components. Commands are sent asynchronously, and the sender of a command is notified when the actions started by the command have completed.

*kgillies@tmt.org; phone: 1-626-395-1655; http://tmt.org

Modeling, Systems Engineering, and Project Management for Astronomy VII,
edited by George Z. Angeli, Philippe Dierickx, Proc. of SPIE Vol. 9911, 99112Q
© 2016 SPIE · CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2234325

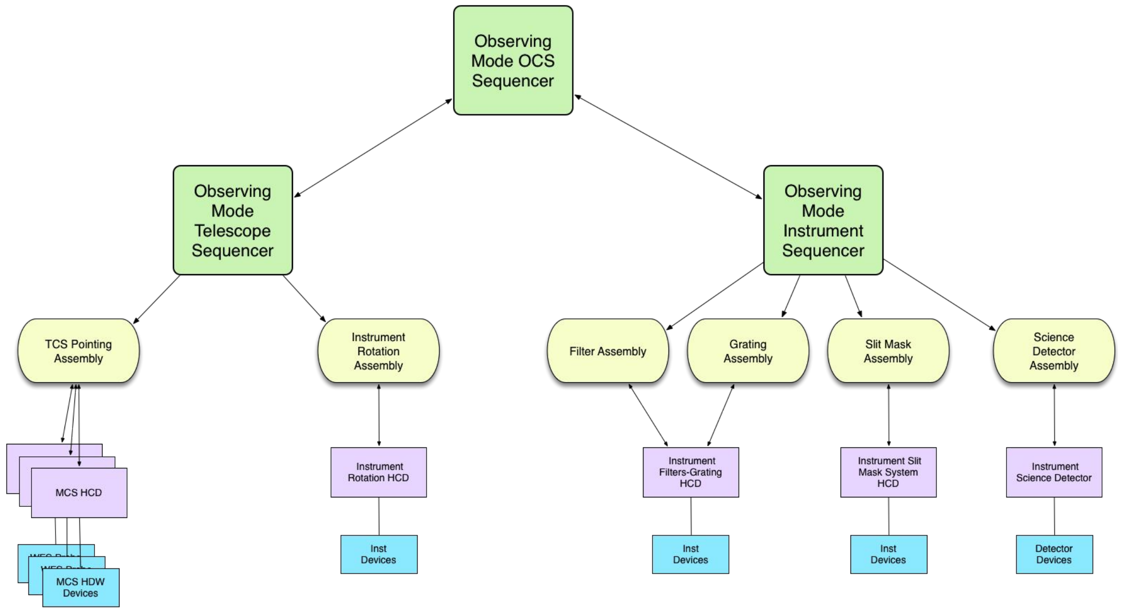

Observing Mode OCS Sequencer
Observing Mode Telescope Sequencer
Observing Mode Instrument Sequencer
TCS Pointing Assembly
Instrument Rotation Assembly
Filter Assembly
Grating Assembly
Slit Mask Assembly
Science Detector Assembly
MCS HCD
Instrument Rotation HCD
Instrument Filters-Grating HCD
Instrument Slit Mask System HCD
Instrument Science Detector
MCS HDW Devices
Inst Devices
Inst Devices
Inst Devices
Detector Devices

Figure 1: An arrangement of CSW standard components consisting of sequencers, assemblies, hardware control daemons, and hardware devices. Arrows indicate hierarchical command flow from sequencers down to hardware.

The second communication service is the Event Service, which allows one component to send a piece of information (i.e., an event) to one or more other components. The service is built using a high-performance, publish-subscribe message system. With the publish-subscribe pattern, components *publish* their information whenever necessary. Other components express interest by *subscribing* to the information and are then notified when new information is published. Subscribers only receive updates to information for which they have subscribed and are updated only when a value is published. The advantage of this type of message system is that publishers and subscribers are decoupled. Publishers can publish regardless of whether there are subscribers, and subscribers can subscribe even if there are no publishers. This is important when systems startup and shutdown independently. The relationship between publishers and subscribers can be one-to-one, one-to-many, many to one, or even many-to-many. Components and systems can startup and stop without requiring the notification of other systems.

There are challenges related to tracking dependencies between components with publish-subscribe style, event-driven interfaces between components. The publish-subscribe pattern allows the creation of event dependencies between systems that are difficult to track but must be managed. Dependencies can be passive or active. When a component subscribes to a topic, but takes no action based on the value it is called a *passive* dependency. For instance, a GUI display could subscribe to the current position of an instrument filter but take no action other than displaying the value.

An *active* dependency occurs when a system takes an action based on a monitored value. For example, an instrument could monitor the telescope zenith angle and modify its internal flexure model based on the published value. If the TCS stopped posting the altitude or posted incorrect values, the instrument would appear to fail in a non-transparent way. Active dependencies are potentially problematic and must be documented for all systems.

2. INTERFACE DATABASE SYSTEM

The Interface Database System (IDBS) project was created to support the documentation of subsystem software components including Application Programming Interface (API) documents and Interface Control Documents (ICD). In addition, the IDBS has the following goals:

- To document the interfaces of components based on Common Software in the TMT Software System. Documentation of software interfaces in traditional ICD documents is tedious, error prone, and documents become obsolete quickly when software changes.
- To understand the software interfaces between subsystems and components in the software system and to understand how events are used in the software system.

- To support a more agile software development process by tracking changes to component interfaces over the course of TMT construction.
- To support the Systems Engineering change control process by understanding how planned interface changes influence the software system.
- To decrease the workload of developers by generating API and ICD documentation that can be used for reviews.

3. IDBS COMPONENTS

The IDBS software consists of two programs called: icd and icd-web, as shown in which are briefly described here and are described in more detail in subsequent sections.

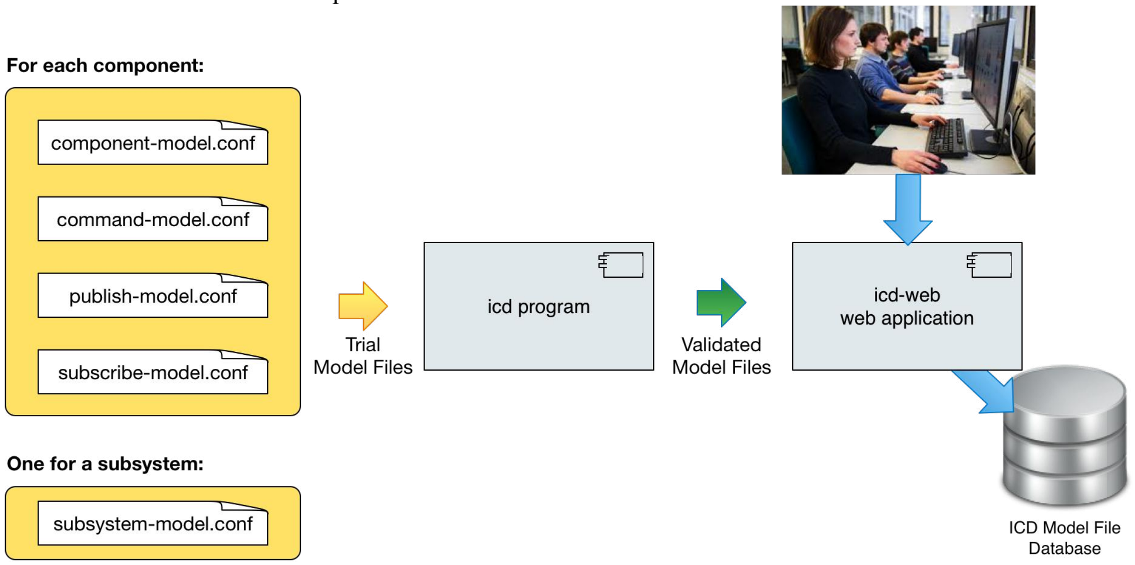


Figure 2: IDBS software includes two programs called icd and icd-web. The icd program is used by programmers to create and validate their model files. The icd-web application provides a user interface and manages the database of model files.

3.1 icd

The *icd* program is used by component developers at their site to create and validate the model files for their components. icd supports the authoring and validating of component model files against a JSON-based model schema. The developer incrementally modifies these files as their components are created and developed. Each pass through the icd program validates the model files or lists errors. The end result is a set of validated model files for each component that can be used to generate documentation or to pass to the next step. The models can be exported as Markdown, HTML, or PDF files at this phase as well.

3.1 icd-web

The icd-web web application provides the IDBS user application interface. The browser-based user interface allows ingesting model files into a model file database, publishing, and displaying versioned “API”s for each component. An API is defined as all the published functionality of a component and subsystem including inputs and outputs and dependencies on components in other subsystems. An ICD is produced by taking the intersection of two published subsystem API documents. ICDs are then also published as versioned documents that depend on published subsystem API versions.

The current version of the IDBS browser user interface includes the following features:

- Browse all the subsystems and their components.
- View the API for a subsystem or an individual component.
- Publish a subsystem API with a version number.
- Browse interfaces between subsystems, or components within subsystems.

- Publish ICD content between two subsystems or a subset of components within subsystems with version information.
- View differences between versions of ICDs or APIs.
- Print ICD or API information from the browser (using browser print).

4. COMPONENT MODELING

The TMT subsystems contain the software components including Hardware Control Daemons, Assemblies, Sequencers, and Applications. The subsystem developer creates model files that provide information about the component, its interfaces, and dependencies. The model files are named: component-model, command-model, publish-model, and subscribe-model. Each subsystem also has a single subsystem-model file. Separate files were created in order to keep the configuration files simpler and easier to type. Table 1 shows the model files and a description of its role and content.

Table 1: Model files and what they describe.

Model File	Description
subsystem-model.conf	Contains high-level information about the subsystem that contains a group of components. There is one subsystem-model.conf file per subsystem.
component-model.conf	Contains high-level component information. There is one component-model.conf file per component.
command-model.conf	Describes the configuration commands the component supports. Also describes components and commands the component may use/send to other components. There is at most one component-model.conf file per component.
publish-model.conf	Describes telemetry/status, events, alarms, and event streams the component publishes using Event Service. There is at most one component-model.conf file per component.
subscribe-model.conf	Describes telemetry/status, events, and event streams of other components the component subscribes to using Event Services. There is at most one component-model.conf file per component.

The component developer is only required to provide a command-model.conf, publish-model.conf, or subscribe-model.conf file if the component provides or uses the features described in the model files (i.e. if the component does not subscribe to events, then a subscribe-model.conf file is not needed).

Model files are text files written in a superset of JSON called HOCON (https://github.com/typesafehub/config/blob/master/HOCON.md), which provides syntactic sugar for writing JSON, but can be exported as pure JSON. The biggest benefit is that quotes are generally not needed. Each model file has a schema, meaning it must follow a certain structure and certain information is required and certain information is optional. This allows the system to validate the model files. Model files can be long, but a brief section of a publish-model.conf file is shown in Figure 3 to show the structure and kind of information that is in a model file. This example shows some telemetry published by the global loop controller (glc) component of the M1CS subsystem.

```
subsystem = M1CS
component = glc

publish {
  telemetry = [
  {
      name = status
      description = """
       |The M1CS shall publish status of the M1CS system once per second.
      """
      requirements = ["INT-TCS-M1CS-2020"]
      minRate = 1
      maxRate = 2
      attributes = [
        {
          name = trackingState
          description = "State of the internal control loop"
          enum = [ HALTED, STANDBY, TRACK ]
        }
        {
          name = outerLoopInputs
          description = "Inputs to the outer loop"
          enum = [NONE, ALL, PTT, FOCUS, ZERNIKE ]
        }
        {
           name = time
           description = "Timestamp associated with the status"
           type = long
        }
      ]
    }
  }
}
```

Figure 3: Model files are written in HOCOM, a superset of JSON. This example shows a telemetry item that is published by the glc component of the M1CS subsystem.

5. WORK FLOW

As shown in Figure 2, model files from the developers are validated and when ready are ingested into the IDBS. The model files define the interface or API of a component including dependencies on other subsystems. The APIs of all the components in a subsystem collectively define the API for the subsystem. An ICD document defines the connections between two subsystems by analyzing the connections between the two subsystems as defined in the model files.

TMT Systems Engineering manages the ICD database and the release of APIs and ICDs from the ICD database as part of the change control process. The entire workflow is shown in Figure 4.

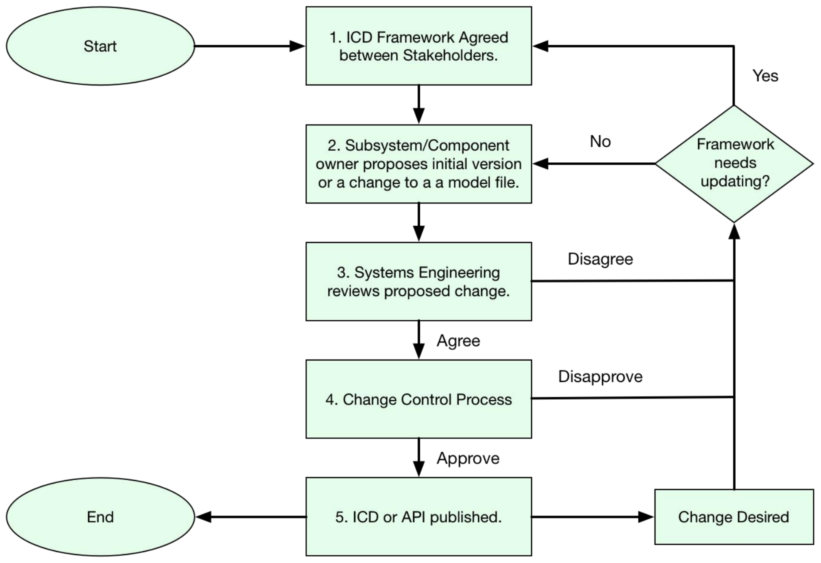
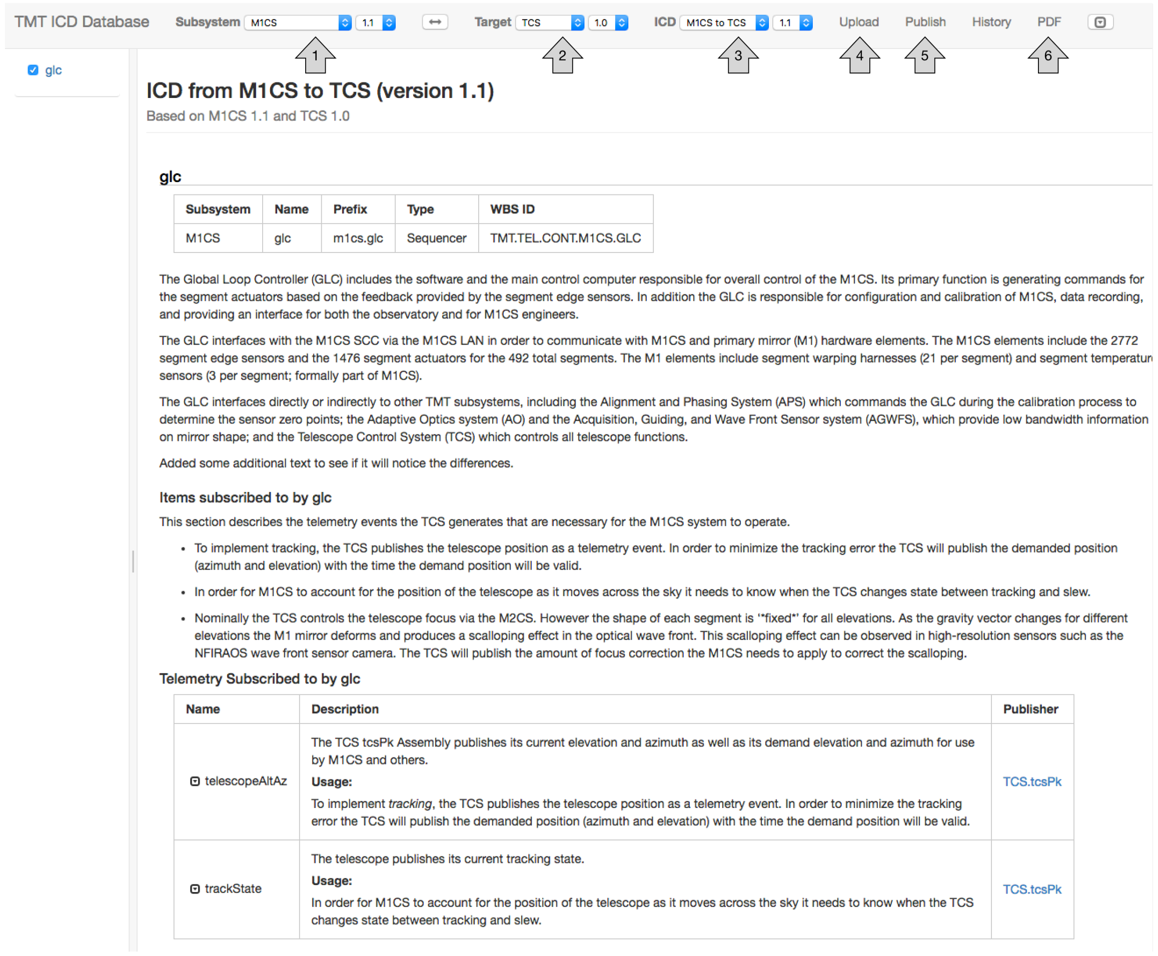

Start
1. ICD Framework Agreed between Stakeholders.
Yes
2. Subsystem/Component owner proposes initial version or a change to a a model file.
No
Framework needs updating?
3. Systems Engineering reviews proposed change.
Disagree
Agree
4. Change Control Process
Disapprove
Approve
End
5. ICD or API published.
Change Desired

Figure 4: The IDBS workflow integrates development of the component model files and the publishing of API and ICD documents into the TMT change control process. The workflow accommodates new ICDs and updates to existing APIs and ICDs.

When creating a new ICD, the stakeholders discuss the interface to decide on the information that is to be passed between subsystems. The interface can consist of commands and/or various types of events. This higher level information is recorded in a narrative framework document, then the detailed interfaces are defined by creating the model files. The producer of data must provide the correct API documentation and the consumer of the data then indicates their use of that information in the model files.

The Change Control process used for software ICDs follows the same basic process as for all TMT technical documentation. The first step in the process is to submit a change request to Systems Engineering. For software ICDs this is done by proposing updates to model files. Systems engineering reviews the changes, and ensures that they are agreed between all stakeholders. The published interfaces consist of the Framework document and the ICD Database report. These two inputs are combined into a single PDF document and placed on the TMT Document Repository. Approval by Systems Engineering, ES&H, and the work package managers is recorded on the document via an electronic routing / approval process.

A software repository has been established on the TMT github site for all subsystem model files (https://github.com/tmtsoftware/TMT-ICD-Model-Files). Each of the 28 TMT subsystems has a directory in the repository that contains the model files for the components of their subsystem. The workflow with git is part of the Systems Engineering change control process. When a subsystem team has an update to their model files, they issue a pull request to the TMT model file repository. This pull request is reviewed by TMT Systems Engineering and either rejected or accepted. When accepted, the changes are merged into the master branch constituting an official release of the subsystem's model files. The git configuration management system provides a record of all changes to the model files. The released set of model files for the observatory are then available to any team member by cloning the model file repository.



ICD from M1CS to TCS (version 1.1)

Based on M1CS 1.1 and TCS 1.0

glc

Subsystem	Name	Prefix	Type	WBS ID
M1CS	glc	m1cs.glc	Sequencer	TMT.TEL.CONT.M1CS.GLC

The Global Loop Controller (GLC) includes the software and the main control computer responsible for overall control of the M1CS. Its primary function is generating commands for the segment actuators based on the feedback provided by the segment edge sensors. In addition the GLC is responsible for configuration and calibration of M1CS, data recording, and providing an interface for both the observatory and for M1CS engineers.

The GLC interfaces with the M1CS SCC via the M1CS LAN in order to communicate with M1CS and primary mirror (M1) hardware elements. The M1CS elements include the 2772 segment edge sensors and the 1476 segment actuators for the 492 total segments. The M1 elements include segment warping harnesses (21 per segment) and segment temperatur sensors (3 per segment; formally part of M1CS).

The GLC interfaces directly or indirectly to other TMT subsystems, including the Alignment and Phasing System (APS) which commands the GLC during the calibration process to determine the sensor zero points; the Adaptive Optics system (AO) and the Acquisition, Guiding, and Wave Front Sensor system (AGWFS), which provide low bandwidth information on mirror shape; and the Telescope Control System (TCS) which controls all telescope functions.

Added some additional text to see if it will notice the differences.

Items subscribed to by glc

This section describes the telemetry events the TCS generates that are necessary for the M1CS system to operate.

- To implement tracking, the TCS publishes the telescope position as a telemetry event. In order to minimize the tracking error the TCS will publish the demanded position (azimuth and elevation) with the time the demand position will be valid.
- In order for M1CS to account for the position of the telescope as it moves across the sky it needs to know when the TCS changes state between tracking and slew.
- Nominally the TCS controls the telescope focus via the M2CS. However the shape of each segment is '"fixed"' for all elevations. As the gravity vector changes for different elevations the M1 mirror deforms and produces a scalloping effect in the optical wave front. This scalloping effect can be observed in high-resolution sensors such as the NFIRAOS wave front sensor camera. The TCS will publish the amount of focus correction the M1CS needs to apply to correct the scalloping.

Telemetry Subscribed to by glc

Name	Description	Publisher
⊡ telescopeAltAz	The TCS tcsPk Assembly publishes its current elevation and azimuth as well as its demand elevation and azimuth for use by M1CS and others. **Usage:** To implement *tracking*, the TCS publishes the telescope position as a telemetry event. In order to minimize the tracking error the TCS will publish the demanded position (azimuth and elevation) with the time the demand position will be valid.	TCS.tcsPk
⊡ trackState	The telescope publishes its current tracking state. **Usage:** In order for M1CS to account for the position of the telescope as it moves across the sky it needs to know when the TCS changes state between tracking and slew.	TCS.tcsPk

Figure 5: Main page of the ICD Database web application showing a published ICD document between the M1CS and TCS subsystems from version 1.1 of the M1CS API and version 1.0 of the TCS API.

6. USER INTERFACE

The icd-web user interface is structured around the operations needed to create and publish an ICD between two subsystems. To create an ICD between the M1CS and TCS subsystems, the following operations are needed:

1. Upload the M1CS component model files.
2. Upload the TCS component model files.
3. Examine and publish the M1CS API with a new version number.
4. Examine and publish the TCS API with a new version number.
5. Create the ICD between the two subsystems by selecting the correct API versions that form the ICD.
6. Examine and publish the ICD between M1CS and ICD.

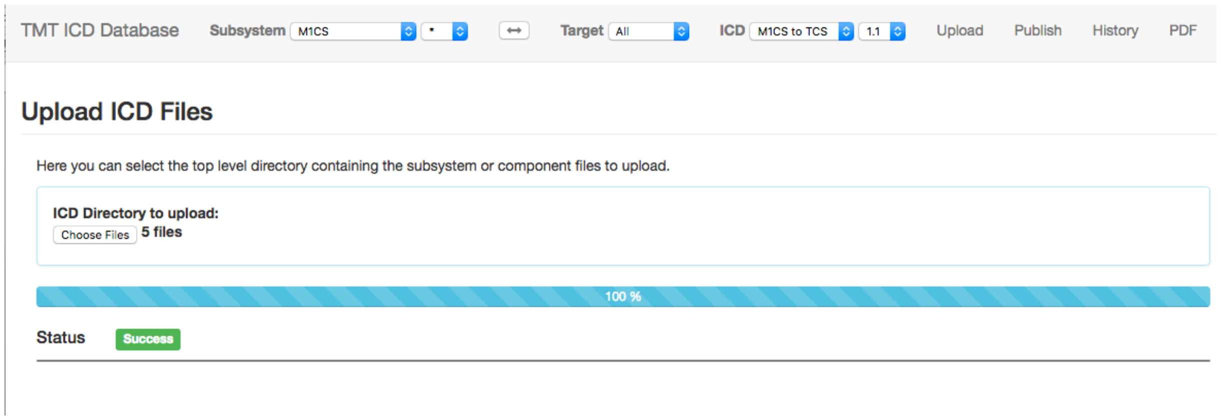
The following sections demonstrate these steps using the web-based user interface. The main page of the icd-web user interface is shown in Figure 5. Across the top is the tool bar area that includes all the application controls.

Arrow 1 points to a selectable menu that selects a subsystem API for viewing. Just to the right is another menu that allows selecting one of the published versions of the API. Arrow 2 points to the selection menu for a second subsystem that is part of an ICD also with a version menu. Currently the user interface is displaying the M1CS to TCS ICD that consists of version 1.1 of the M1CS API and version 1.0 of the TCS API. The text of the selected output ICD (or API) is shown in the center area, which changes based on tool bar selections. At the left is a list of the components that are included in the selected subsystem. The checkboxes can be used to filter the displayed content.

Arrow 3 points to a selectable menu of published ICDs and their version numbers. As configured in Figure 5, version 1.1 of the M1CS to TCS ICD is selected. Arrow 4 is the button used to upload model files. Arrow 5 points to a button used to publish an API or ICD that has been configured. Arrow 6 points to a button that can be used to create a PDF format API or ICD document for printing or storage.

6.1 Uploading model files

A TMT software subsystem may have many components, and each will have a set of model files that describe its API. The model files are edited and verified using the icd program and are then ready to be uploaded to the database using the icd-web user interface. The user selects the upload button (arrow 4 in Figure 5) and the page and form in Figure 6 is shown.



Subsystem: M1CS *
Target: All
ICD: M1CS to TCS 1.1

Upload ICD Files

Here you can select the top level directory containing the subsystem or component files to upload.

ICD Directory to upload:
Choose Files 5 files

100 %

Status Success

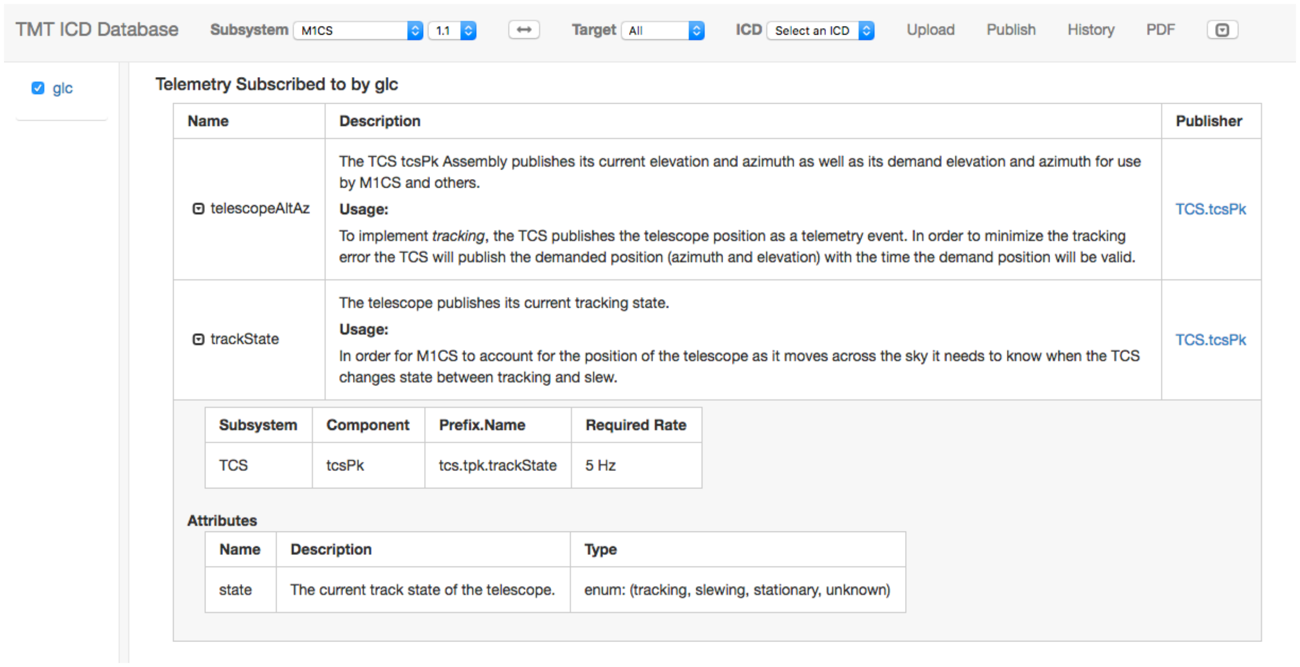
Figure 6: The Upload button is pushed allowing the selection of an individual model file or a directory containing one or more sets of component model files.

The user chooses the file or directory containing one or more sets of component files and all are uploaded, verified and stored into the database as working copies that are not yet published. Until they are published, they can be modified and uploaded over and over.

6.2 Examine and publish an API

The API for a subsystem is displayed in the content area by selecting the subsystem and version (arrow 1 in Figure 5). This is a useful online feature for programmers who need to browse the available functionality of a subsystem or component and can be used during the exploratory stages of developing an ICD. Figure 7 is a section of the version 1.1 M1CS API that shows the telemetry events subscribed to by M1CS.glc: telescopeAltAz and trackState. Each item includes a description and text indicating how the subscriber uses the telemetry item. The trackState shows the expanded more detailed view of the published telemetry event. All this information comes from the model files.

The publisher column on the right is a link to the documentation for the TCS tcsPk component that publishes the telemetry item. When the link is selected, the tcsPk component is selected in the content area as shown in Figure 8. This now shows all the telemetry events published by tcsPk. The rightmost column shows the subscribers to each event. As shown, the zenithAngle event has three subscribers.



Telemetry Subscribed to by glc

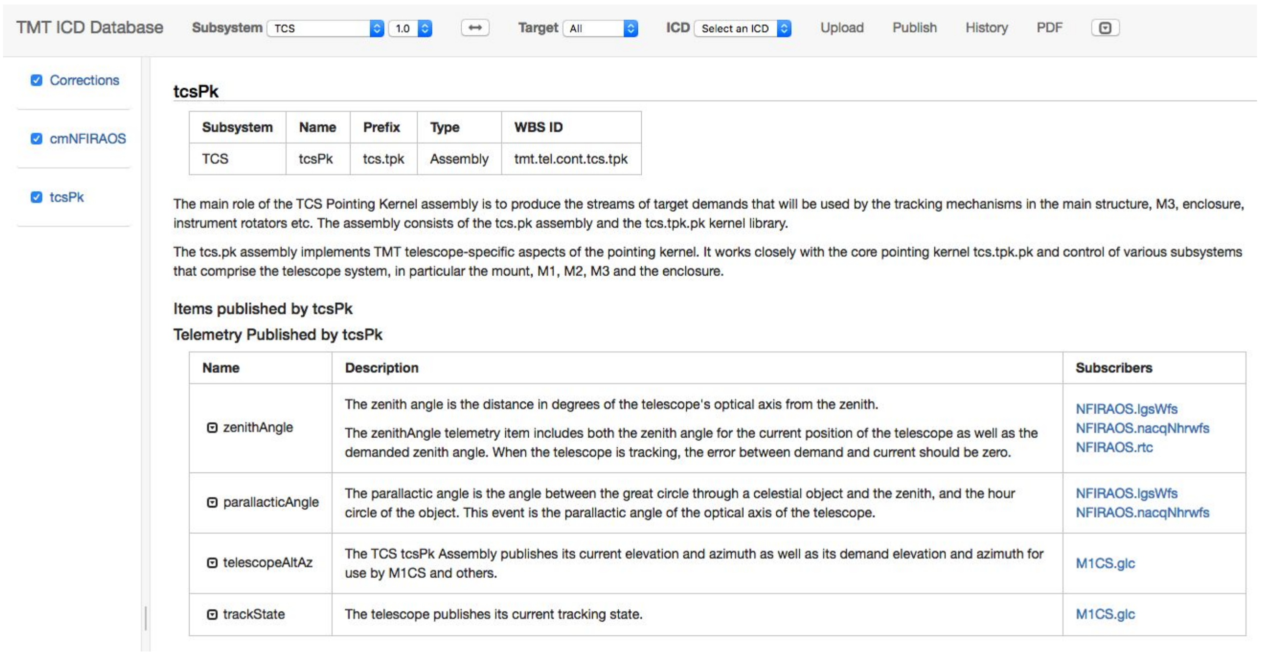
Name	Description	Publisher
telescopeAltAz	The TCS tcsPk Assembly publishes its current elevation and azimuth as well as its demand elevation and azimuth for use by M1CS and others. **Usage:** To implement *tracking*, the TCS publishes the telescope position as a telemetry event. In order to minimize the tracking error the TCS will publish the demanded position (azimuth and elevation) with the time the demand position will be valid.	TCS.tcsPk
trackState	The telescope publishes its current tracking state. **Usage:** In order for M1CS to account for the position of the telescope as it moves across the sky it needs to know when the TCS changes state between tracking and slew.	TCS.tcsPk

Subsystem	Component	Prefix.Name	Required Rate
TCS	tcsPk	tcs.tpk.trackState	5 Hz

Attributes

Name	Description	Type
state	The current track state of the telescope.	enum: (tracking, slewing, stationary, unknown)

Figure 7: The M1CS API is displayed showing telemetry used by the glc component of M1CS that is published by the TCS.tcsPk component.



tcsPk

Subsystem	Name	Prefix	Type	WBS ID
TCS	tcsPk	tcs.tpk	Assembly	tmt.tel.cont.tcs.tpk

The main role of the TCS Pointing Kernel assembly is to produce the streams of target demands that will be used by the tracking mechanisms in the main structure, M3, enclosure, instrument rotators etc. The assembly consists of the tcs.pk assembly and the tcs.tpk.pk kernel library.

The tcs.pk assembly implements TMT telescope-specific aspects of the pointing kernel. It works closely with the core pointing kernel tcs.tpk.pk and control of various subsystems that comprise the telescope system, in particular the mount, M1, M2, M3 and the enclosure.

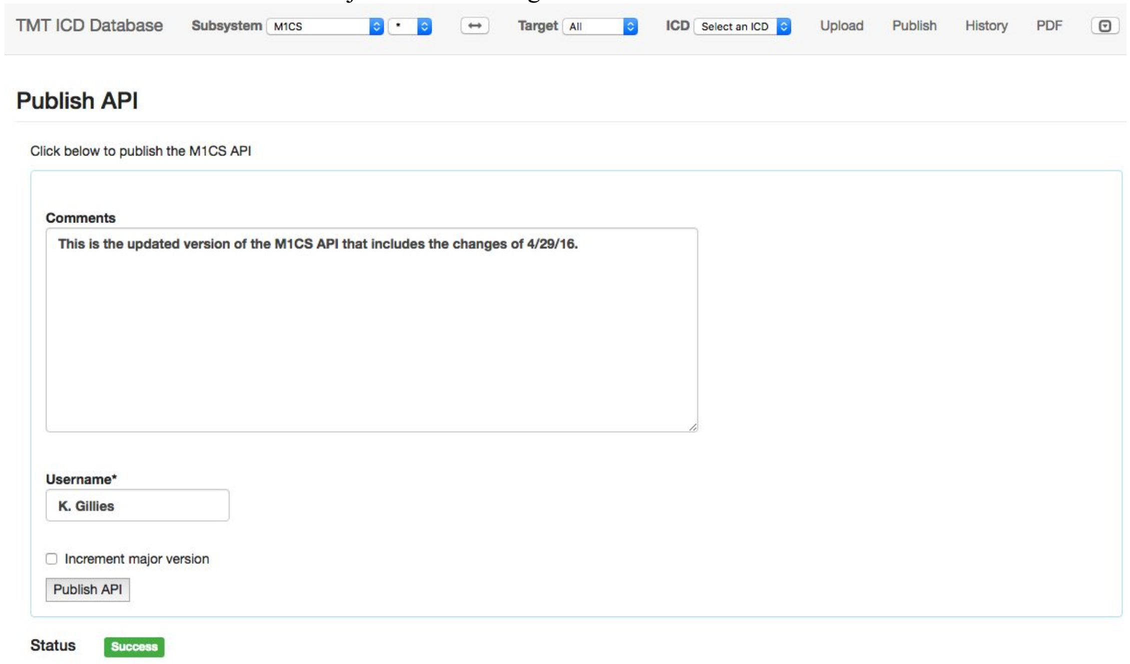
Items published by tcsPk

Telemetry Published by tcsPk

Name	Description	Subscribers
zenithAngle	The zenith angle is the distance in degrees of the telescope's optical axis from the zenith. The zenithAngle telemetry item includes both the zenith angle for the current position of the telescope as well as the demanded zenith angle. When the telescope is tracking, the error between demand and current should be zero.	NFIRAOS.lgsWfs NFIRAOS.nacqNhrwfs NFIRAOS.rtc
parallacticAngle	The parallactic angle is the angle between the great circle through a celestial object and the zenith, and the hour circle of the object. This event is the parallactic angle of the optical axis of the telescope.	NFIRAOS.lgsWfs NFIRAOS.nacqNhrwfs
telescopeAltAz	The TCS tcsPk Assembly publishes its current elevation and azimuth as well as its demand elevation and azimuth for use by M1CS and others.	M1CS.glc
trackState	The telescope publishes its current tracking state.	M1CS.glc

Figure 8: The TCS tcsPk API shows telemetry items published by TCS and a list of all known subscribers to the right. This is used to understand the connections between components in the system.

When satisfied that the version of the API is correct, it can be published by selecting the publish button (arrow 5 in Figure 5) with the result shown in Figure 9. The name of the user and a comment are added to be stored with the record of published APIs when the publish button is selected. The new API version is given a new version number by the system, but the user can also select a major revision change when needed.



Publish API

Click below to publish the M1CS API

Comments

This is the updated version of the M1CS API that includes the changes of 4/29/16.

Username*

K. Gillies

☐ Increment major version

Publish API

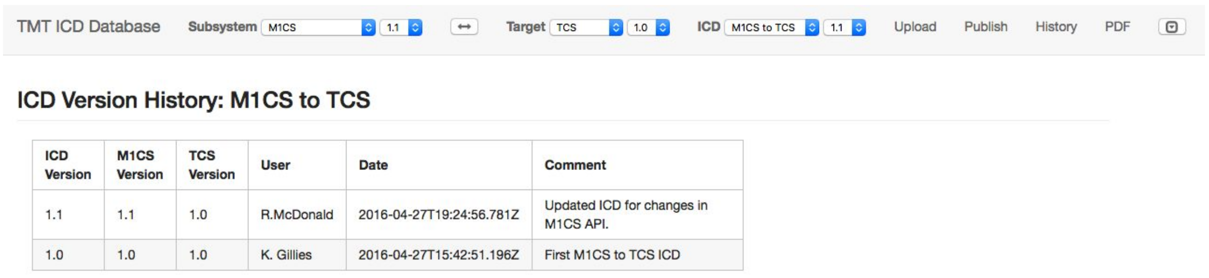
Status Success

Figure 9: When publishing an API version, the name of the user and a comment is added to the document history.

6.3 Examine and publish an ICD

An ICD exists between two subsystems. In order to create an ICD, the two subsystems must have published an API and the two API versions are selected in the subsystem and target menus. The application examines the two APIs and looks for the intersection of the model file information to produce the ICD. Any API information that is not part of the interface between the two subsystems is not displayed in the ICD. Figure 5 shows an ICD between the M1CS and TCS subsystems. As shown it is formed from the intersection of versions 1.1 of the M1CS API and 1.0 of the TCS API. As with an API, the ICD is a working version until it is published. After selecting the two API versions, the publish button (arrow 5 in Figure 5) is selected and a form similar to Figure 9 is shown allowing the user to enter a comment and name. The system stores all information about published APIs and ICDs.

This is shown in Figure 10, which shows the history for the M1CS to TCS ICD. The table includes the API versions that are used to form each ICD version along with the name of the user that published the document and a comment for each version.

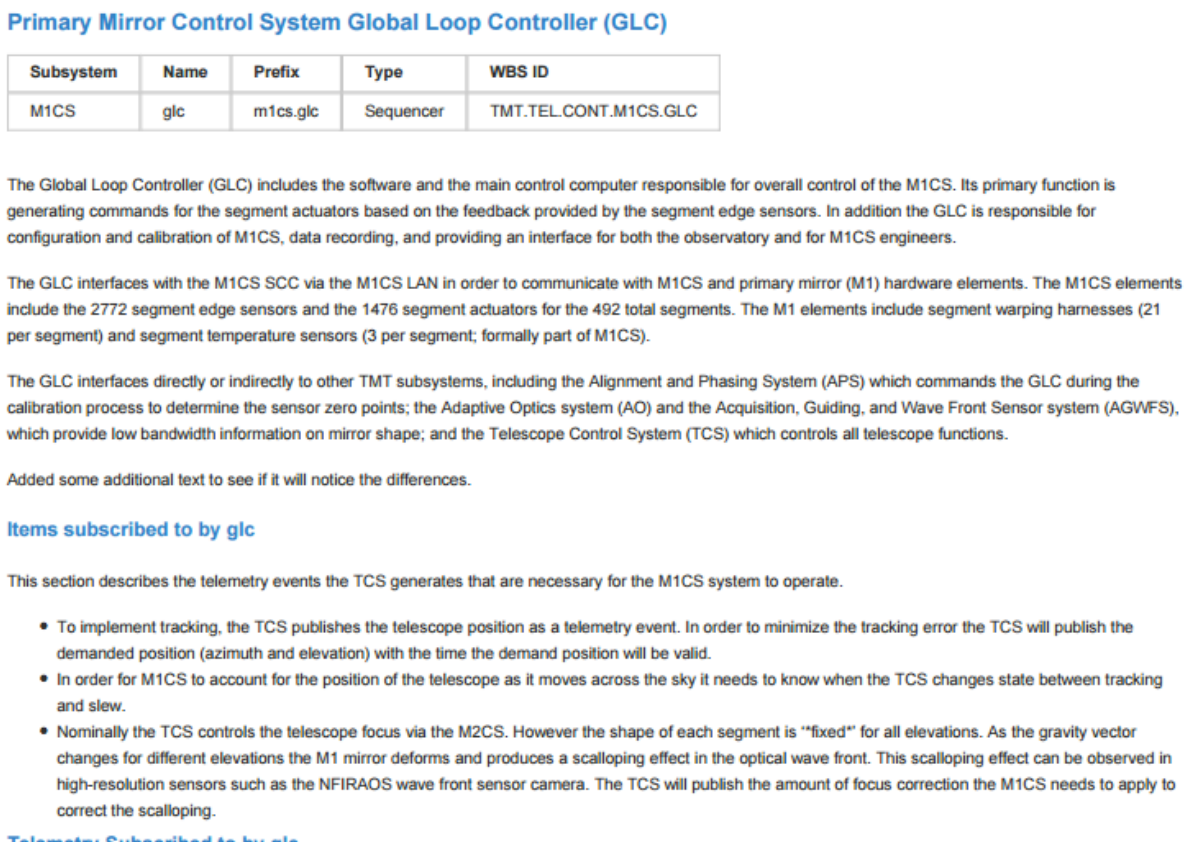


ICD Version History: M1CS to TCS

ICD Version	M1CS Version	TCS Version	User	Date	Comment
1.1	1.1	1.0	R.McDonald	2016-04-27T19:24:56.781Z	Updated ICD for changes in M1CS API.
1.0	1.0	1.0	K. Gillies	2016-04-27T15:42:51.196Z	First M1CS to TCS ICD

Figure 10: The IDBS stores information for each published API and ICD version including the name of the user who published the document and their comment.

Finally, when a hard copy of an ICD is needed, IDBS can generate a PDF formatted document that can be saved or printed. An example of the PDF output for the M1CS to TCS ICD is shown in Figure 11. This might be appropriate for a formal review.



Primary Mirror Control System Global Loop Controller (GLC)

Subsystem	Name	Prefix	Type	WBS ID
M1CS	glc	m1cs.glc	Sequencer	TMT.TEL.CONT.M1CS.GLC

The Global Loop Controller (GLC) includes the software and the main control computer responsible for overall control of the M1CS. Its primary function is generating commands for the segment actuators based on the feedback provided by the segment edge sensors. In addition the GLC is responsible for configuration and calibration of M1CS, data recording, and providing an interface for both the observatory and for M1CS engineers.

The GLC interfaces with the M1CS SCC via the M1CS LAN in order to communicate with M1CS and primary mirror (M1) hardware elements. The M1CS elements include the 2772 segment edge sensors and the 1476 segment actuators for the 492 total segments. The M1 elements include segment warping harnesses (21 per segment) and segment temperature sensors (3 per segment; formally part of M1CS).

The GLC interfaces directly or indirectly to other TMT subsystems, including the Alignment and Phasing System (APS) which commands the GLC during the calibration process to determine the sensor zero points; the Adaptive Optics system (AO) and the Acquisition, Guiding, and Wave Front Sensor system (AGWFS), which provide low bandwidth information on mirror shape; and the Telescope Control System (TCS) which controls all telescope functions.

Added some additional text to see if it will notice the differences.

Items subscribed to by glc

This section describes the telemetry events the TCS generates that are necessary for the M1CS system to operate.

- To implement tracking, the TCS publishes the telescope position as a telemetry event. In order to minimize the tracking error the TCS will publish the demanded position (azimuth and elevation) with the time the demand position will be valid.
- In order for M1CS to account for the position of the telescope as it moves across the sky it needs to know when the TCS changes state between tracking and slew.
- Nominally the TCS controls the telescope focus via the M2CS. However the shape of each segment is "fixed" for all elevations. As the gravity vector changes for different elevations the M1 mirror deforms and produces a scalloping effect in the optical wave front. This scalloping effect can be observed in high-resolution sensors such as the NFIRAOS wave front sensor camera. The TCS will publish the amount of focus correction the M1CS needs to apply to correct the scalloping.

Figure 11: A PDF file can be generated for an ICD or API document. Shown is a section of the M1CS to TCS ICD. The PDF file can be stored or printed.

7. IMPLEMENTATION

The IDBS is implemented in Scala, a JVM-based language similar to Java (http://www.scala-lang.org). The web application is developed using the Play web application framework (https://www.playframework.com) and scala.js for the user interface (https://www.scala-js.org). These are some of the tools selected for use in the TMT software system. The JSON model files are stored as binary files in a document database system called MongoDB (https://www.mongodb.org). The IDBS code is open source and is available in the TMT github at: https://github.com/tmtsoftware/icd. While the IDBS is somewhat specific to the TMT Common Software, it may possibly be used in other similar systems with a few changes. The concepts of commands and events are often used in observatory systems.

8. SUMMARY

The Interface Database System was created to help solve a problem that exists in the distributed TMT Software System where software components developed by different groups are dependent on one another in ways that are difficult to understand and track. The IDBS allows these connections to be modeled and as a result they can be managed. The connections and dependencies are more transparent and changes to software interfaces can be tracked through traditional system engineering processes. A side effect of this is a system that allows developers to browse the capabilities of the system and produce painful documentation. There is great untapped potential in this system, and we hope to continue to develop more uses in future years.

ACKNOWLEDGEMENTS

The TMT Project gratefully acknowledges the support of the TMT collaborating institutions. They are the California Institute of Technology, the University of California, the National Astronomical Observatory of Japan, the National Astronomical Observatories of China and their consortium partners, the Department of Science and Technology of India and their supported institutes, and the National Research Council of Canada. This work was supported as well by the Gordon and Betty Moore Foundation, the Canada Foundation for Innovation, the Ontario Ministry of Research and Innovation, the Natural Sciences and Engineering Research Council of Canada, the British Columbia Knowledge Development Fund, the Association of Canadian Universities for Research in Astronomy (ACURA), the Association of Universities for Research in Astronomy (AURA), the U.S. National Science Foundation, the National Institutes of Natural Sciences of Japan, and the Department of Atomic Energy of India.