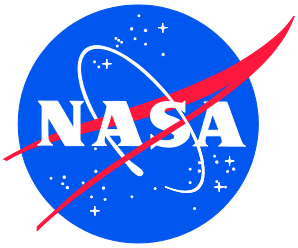
NASA/TM–2019-220247
NESC-RP-15-01097




# Improvements to the Copernicus Trajectory Design and Optimization System for Complex Space Trajectories

*Daniel G. Murri/NESC*
*Langley Research Center, Hampton, Virginia*

*Gerald L. Condon*
*Johnson Space Center, Houston, Texas*

*Jacob Williams and Anubhav H. Kamath*
*Jacobs Technology, Houston, Texas*

*Randy A. Eckman*
*Johnson Space Center, Houston, Texas*

*Ravishankar Mathur*
*Emergent Space Technologies, Laurel, Maryland*

January 2019

**NASA STI Program . . . in Profile**

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

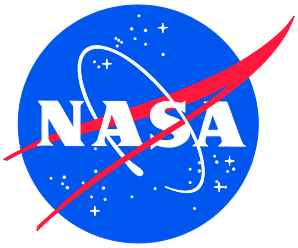
- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM–2019-220247
NESC-RP-15-01097




# Improvements to the Copernicus Trajectory Design and Optimization System for Complex Space Trajectories

*Daniel G. Murri/NESC*
*Langley Research Center, Hampton, Virginia*

*Gerald L. Condon*
*Johnson Space Center, Houston, Texas*

*Jacob Williams and Anubhav H. Kamath*
*Jacobs Technology, Houston, Texas*

*Randy A. Eckman*
*Johnson Space Center, Houston, Texas*

*Ravishankar Mathur*
*Emergent Space Technologies, Laurel, Maryland*

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

January 2019

**Acknowledgments**

The authors wish to acknowledge those who have beta-tested the Copernicus version 5.0 release: David E. Lee at the Johnson Space Center (JSC), Amelia Batcha (JSC), Timothy Dawn (JSC), and Elizabeth Williams (JSC/, a.i. Solutions).



Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

# NASA Engineering and Safety Center
# Technical Assessment Report

# Improvements to the Copernicus Trajectory Design and Optimization System for Complex Space Trajectories

**November 15, 2018**

# Report Approval and Revision History

NOTE: This document was approved at the November 15, 2018, NRB. This document was submitted to the NESC Director on November 27, 2018, for configuration control.

| | | |
|---|---|---|
| Approved: | *Original Signature on File* | 11/27/18 |
| | NESC Director | Date |

| Version | Description of Revision | Office of Primary Responsibility | Effective Date |
|---|---|---|---|
| 1.0 | Initial Release | Mr. Daniel Murri, NASA Technical Fellow for Flight Mechanics | 11/15/18 |

# Table of Contents

## Technical Assessment Report

## List of Figures

# Technical Assessment Report

## 1.0 Notification and Authorization

The purpose of this assessment was to develop updates and new features for the NASA Copernicus Spacecraft Trajectory Design and Optimization analysis tool (version 5.0) for application to NASA programs and projects. These updates will significantly improve the ability to design and optimize complex trajectories over multiple trajectory phases; will allow the use of unique vehicle-specific guidance, control, and trajectory strategies and constraints; and the creation of an almost unlimited number of unique user-defined capabilities. The primary stakeholders for this assessment are the trajectory design and optimization analysts and engineers, and the chief engineers and project managers for existing programs, projects, and/or tasks that involve impulsive, finite burn, and/or continuous thrust trajectories (e.g., Sun, planet, comet, asteroid, halo orbit, Lagrange point, and distant retrograde orbit). The breadth of application spans the preliminary engineering and mission design concepts and optimization, to the development of candidate reference missions and integrated mission design for vehicle system design and operation, to the design and development of flight trajectories and associated propulsive maneuvers for real-time operations.

## 2.0 Signature Page

Submitted by:

*Team Signature Page on File – 11/28/18*

Mr. Daniel G. Murri Date

Significant Contributors:

Mr. Gerald L. Condon Date

Mr. Jacob Williams Date

Mr. Anu H. Kamath Date

Mr. Randy A. Eckman Date

Dr. Ravi Mathur Date

Signatories declare the findings, observations, and NESC recommendations compiled in the report are factually based from data extracted from program/project documents, contractor reports, and open literature, and/or generated from independently conducted tests, analyses, and inspections.

# 3.0 Team List

| Name | Discipline | Organization |
|---|---|---|
| **Core Team** | | |
| Daniel Murri | NESC Lead | LaRC |
| Jerry Condon | Copernicus Development Team Lead | JSC |
| Jacob Williams | Copernicus Lead Developer | JSC/JETS |
| Laura Burke | Copernicus Development Support | GRC |
| Randy Eckman | Copernicus Developer | JSC |
| Anu Kamath | Copernicus Developer | JSC/JETS |
| Melissa McGuire | Copernicus Development Support | GRC |
| Ravi Mathur | OpenFrames Developer | GSFC/Emergent |
| Matthew Ruschmann | OpenFrames Developer | GSFC/Emergent |
| Juan Senent | Mission Design Analyst, Plug-in Support | JPL |
| Mark Jesick | Mission Design Analyst, Plug-in Support | JPL |
| **Consultants** | | |
| Cesar Ocampo | Copernicus Creator, Trajectory Optimization Expert | JSC/Odyssey |
| Joseph Guinn | Integration with Monte | JPL |
| **Business Management** | | |
| John LaNeave | Program Analyst | LaRC/MTSO |
| **Assessment Support** | | |
| Linda Burgess | Planning and Control Analyst | LaRC/AMA |
| Melinda Meredith | Project Coordinator | LaRC/AMA |
| Erin Moran | Technical Editor | LaRC/AMA |

## 3.1 Acknowledgements

The authors wish to acknowledge those who have beta-tested the Copernicus version 5.0 release: David E. Lee at the Johnson Space Center (JSC), Amelia Batcha (JSC), Timothy Dawn (JSC), and Elizabeth Williams (JSC/, a.i. Solutions).

## 4.0 Executive Summary

NASA's Copernicus spacecraft trajectory optimization and design analysis tool is a comprehensive and generalized spacecraft trajectory design and optimization system. It forms part of the suite of tools used by NASA, industry, and academia to study, design, and execute spacecraft missions. It is intended to be a tool that evolves and conforms to current trends and requirements associated with spacecraft trajectory optimization, design, and operation. Copernicus is capable of solving a wide range of trajectory design and optimization problems. These include trajectories centered about any celestial body or location in the solar system as well as trajectories influenced by two or more bodies. Examples include: orbit-to-orbit transfers about a given planet (or moon or asteroid, etc.), orbit to hyperbolic departure from a given body, libration point trajectories/halo orbits, distant retrograde orbits, frozen orbits, other restricted three body model trajectories, Earth-Moon and interplanetary transfers, asteroid and comet missions, etc. At the NASA Johnson Space Center (JSC), Copernicus is the primary trajectory optimization tool used for the design of integrated Space Launch System (SLS)/Orion Multi-Program Crew Vehicle (MPCV) missions in the current NASA Exploration Mission (EM) launch series.

This report details significant upgrades that were made to Copernicus in support of the NESC assessment. These upgrades represent major new capabilities that have been added to the tool for support of a variety of NASA projects and missions. The tool is more powerful, versatile, and user friendly, and is built on a modern graphical user interface (GUI) toolkit (i.e., PyQt).

This work has resulted in major improvements and new capabilities for the Copernicus tool including, but not limited to: improvements to the OpenFrames library, used to provide three-dimensional (3D) OpenGL graphics, visualization capability; a NASA-developed GUI; a cross-platform set of Copernicus versions that run natively on PC, Mac, or Linux operating systems; an integrated Python scripting environment for manipulation of individual Copernicus input decks and automated external assessment and response for active Copernicus runs; an enhanced plugin technology allowing Copernicus to incorporate unique existing or user developed algorithms; and a host of architectural modifications and improvements designed to provide a faster, more user-friendly, more capable experience.

The result of these improvements and capabilities has been multi-faceted. Current primary NASA programs (i.e., SLS, Orion MPCV) have seen the blossoming of not only single mission trajectory optimization capability, but also ancillary benefits (e.g., easier to implement, yet more sophisticated, application to large trajectory scans for design trade studies, and abort space assessment of established reference mission trajectories).

The continuing enhancement of the visualization has made Copernicus a more visually immersive tool, particularly for initial design, design modification, trouble-shooting, and exploration of possible new trajectory options. In some cases, users have been able to visually construct complex trajectories (e.g., distant retrograde orbit (DRO) with a propagated orbit lifetime of many decades without need for orbit maintenance).

The OpenFrames Application Programming Interface (API) allows developers to embed real-time 3D interactive visualizations into their simulations. The Copernicus trajectory design and optimization tool has used OpenFrames for its visualizations for over a decade. During this assessment, new features and enhancements to OpenFrames were developed that directly benefit Copernicus. Examples include support for advanced user interfaces embedded in the 3D scene,

realistic lighting on celestial bodies and spacecraft, hyper-realistic celestial body models that increase resolution as the viewer approaches the surface, viewing a scene in consumer-grade virtual reality (VR) hardware (e.g., Oculus Rift or HTC[1] Vive), and displaying visualizations in Copernicus' PyQt-based GUI. Furthermore, because OpenFrames is Open Source Software (OSS), the advancements made will benefit other NASA applications that use OpenFrames by increasing software reuse and reducing software development costs. Examples include the General Mission Analysis Tool (GMAT) and the Virtual Landscapes VR science exploration tool.

The PyQt GUI has been a significant boost to user efficiency, environment customization, and situational awareness by providing streamlined workflow and product development. Additionally, NASA developed this GUI so it has full control over its design, update, and modification.

The Copernicus version 5.0 improvements have advanced the capability of a single user in the speed and accuracy of a mission design and the volume of data product that a single user can generate. This has enhanced the potential and realized trajectory design/optimization capability of the individual user and allows projects and programs to accomplish mission planning and analysis with fewer people, and/or expand the scope and depth of analysis for key mission related program decision-making (e.g., assessment of the scope of abort capability available to a given reference mission).

In summary, the Copernicus version 5.0 development activity produced significant improvements in several key areas of tool design and function, which include: improved visualization, a PyQt-based GUI, expanded and updated Python scripting language implementation, an enhanced plug-in implementation, and other improvements. The task produced three independent platform versions (i.e., PC, Mac, and Linux) that run natively on each respective platform. In addition, the assessment team was able to collaborate with the Small Business and Innovative Research (SBIR) Program on two topic areas to obtain additional visualization development beyond the scope of the original task.

[1] HTC is a company that makes the Vive model virtual reality headset.

# 5.0 Assessment Plan

The main purpose of the initial assessment plan was to develop updates and new features for the NASA Copernicus Trajectory Design and Optimization tool for application to NASA programs and projects. The goal was to include numerous improvements to provide enhanced mission optimization/performance, and reduced mission risk for human and science mission spaceflight programs. The assessment included five major technical items:

1. Mac-based version of Copernicus
2. Linux-based version of Copernicus
3. Plug-ins and scripting improvements
4. GUI and 3D graphics visualization improvements
5. Copernicus software architecture improvements

Each item represented a significant upgrade or set of upgrades to the Copernicus tool. All major items were completed (details are discussed in Section 6.0). The implementation details of some items varied somewhat from how they were originally envisioned in the assessment plan. For example, it was originally envisioned that the new cross-platform (i.e., Mac and Linux) builds of Copernicus would be accomplished using the existing Winteracter-based toolkit that Copernicus has used since its initial 1.0 release in 2006. However, upon further investigation, it was decided that the best path forward, for the various reasons discussed in the following section, was to replace this toolkit with an open source toolkit (i.e., PyQt). This proved to provide significant advantages that would not have been possible using the existing toolkit.

# 6.0 Copernicus Enhancements

A summary of Copernicus enhancements developed by this assessment are discussed in this section. This updated version of Copernicus is version 5.0. As part of the development process for version 5.0, some of the updates were distributed in versions 4.5 and 4.6. The current release of Copernicus (version 4.6) is available at JSC Tech Transfer link, https://software.nasa.gov/software/MSC-25863-1. The updated version 5.0 will be available at this link once beta testing is completed. At any time, the latest version of Copernicus can be obtained by going to this website and clicking on "Request Now!"

## 6.1 New Copernicus GUI

### Background

Until version 4.6, Copernicus used a Fortran GUI toolkit Winteracter. However, as the capabilities of Copernicus has increased, it has outgrown the available capabilities of Winteracter, and a change to the PyQt GUI has provided the needed new capability. This section outlines some of the benefits of the PyQt GUI.

## 6.2 Benefits of the PyQt GUI

### GUI can be modified internally:

The Copernicus version 5.0 GUI was developed using open source Qt and PyQt. Since the GUI source code was developed by NASA, there is freedom to modify the GUI and a rapid resolution on bug fixes and incorporation of new features when needed. Previously, if there was a bug or

feature that was needed inherent to the GUI, the Copernicus developers had to rely on Winteracter developers to implement a change. In addition, the Python source code for the GUI will be released with Copernicus, providing users the ability to modify it, if necessary or desired.

**Decoupling of core capability and GUI logic:**

The GUI code is contained within a set of Python files and is segregated from the core Fortran code of Copernicus. The Python and Fortran code interact via an API and callback interface used to pass data back and forth. Previously, the Winteracter API was collocated within the Fortran code. This was not ideal because the separation of the GUI and core logic was often unclear. Another benefit of separating the two code bases is that defects that pertain only to the GUI do not require a Fortran rebuild. It is possible to release patches to fix GUI issues without users having to fully upgrade the Copernicus software.

Another benefit is that by using the API documentation, a completely different GUI using something other than PyQt could be developed and used in the future without affecting the Fortran code. This allows for the realization of potential future concepts, such as web- or mobile app-based GUI.

**Improved usability and layout:**

From the outset, the PyQt GUI looks and feels more modern (Figures 6.2-1 through 6.2-5). The user is able to customize the theme, font and size, layout, and has access to features that were not available in Winteracter. The following is a list of features that are improved or were unavailable in Winteracter:

- Highly customizable to user preferences (e.g., themes, fonts, undockable/hideable widgets, layout of widgets, etc.). A widget is a useful tool or device, including, in the case of a GUI, a radio button, a slider, a text field, etc.
- Full screen graphics window. In addition, the user can split the GUI window so that the main GUI is on one monitor, and the 3D graphics are on another monitor.
- Reduced number of dialogs (e.g., previously a certain action needed 3 dialogs, it can now be done in 2).
- New usability features (e.g., dialog tabbing, click and drag capability, easy access toolbar buttons and shortcuts, search tool, and undo/redo in certain dialogs).

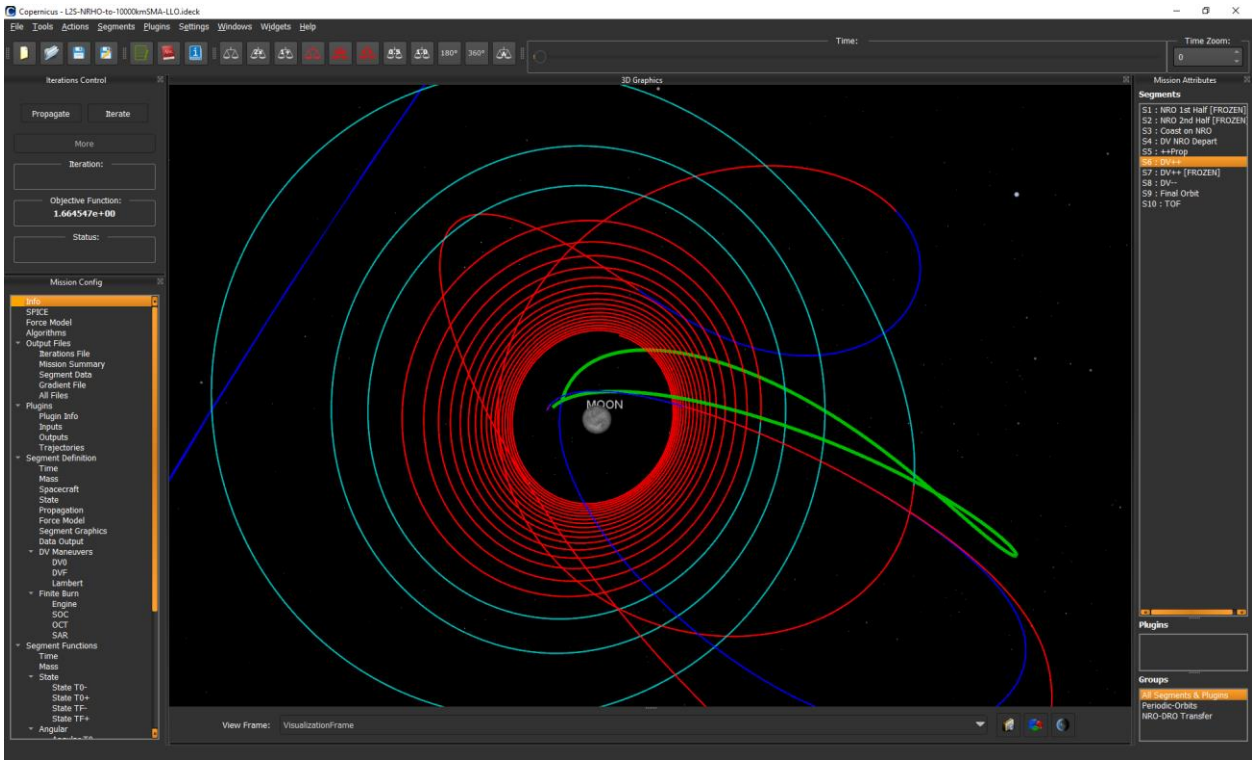



***Figure 6.2-1. PyQt Copernicus GUI***

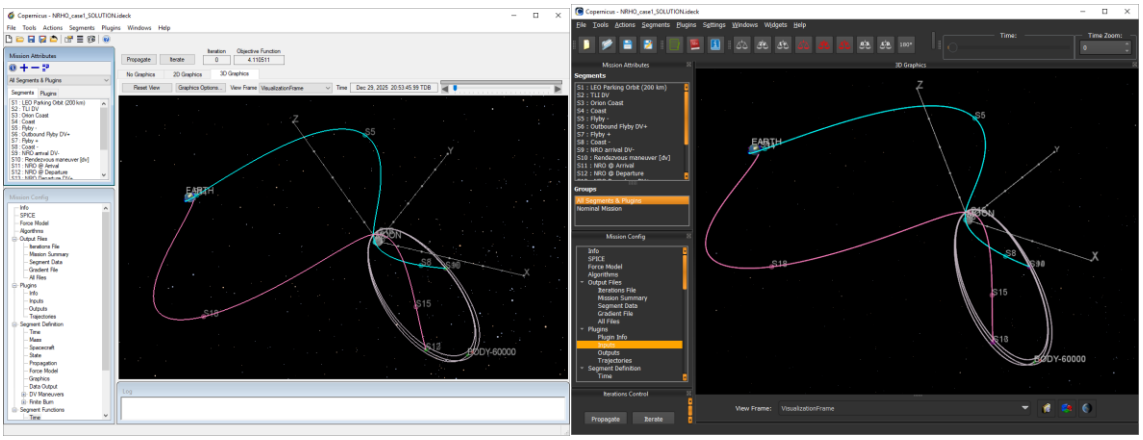



***Figure 6.2-2. Old vs New GUI. Users of the old GUI (left) will find the new GUI (right) very familiar. The new GUI is much more configurable, whereas the old GUI was very static.***

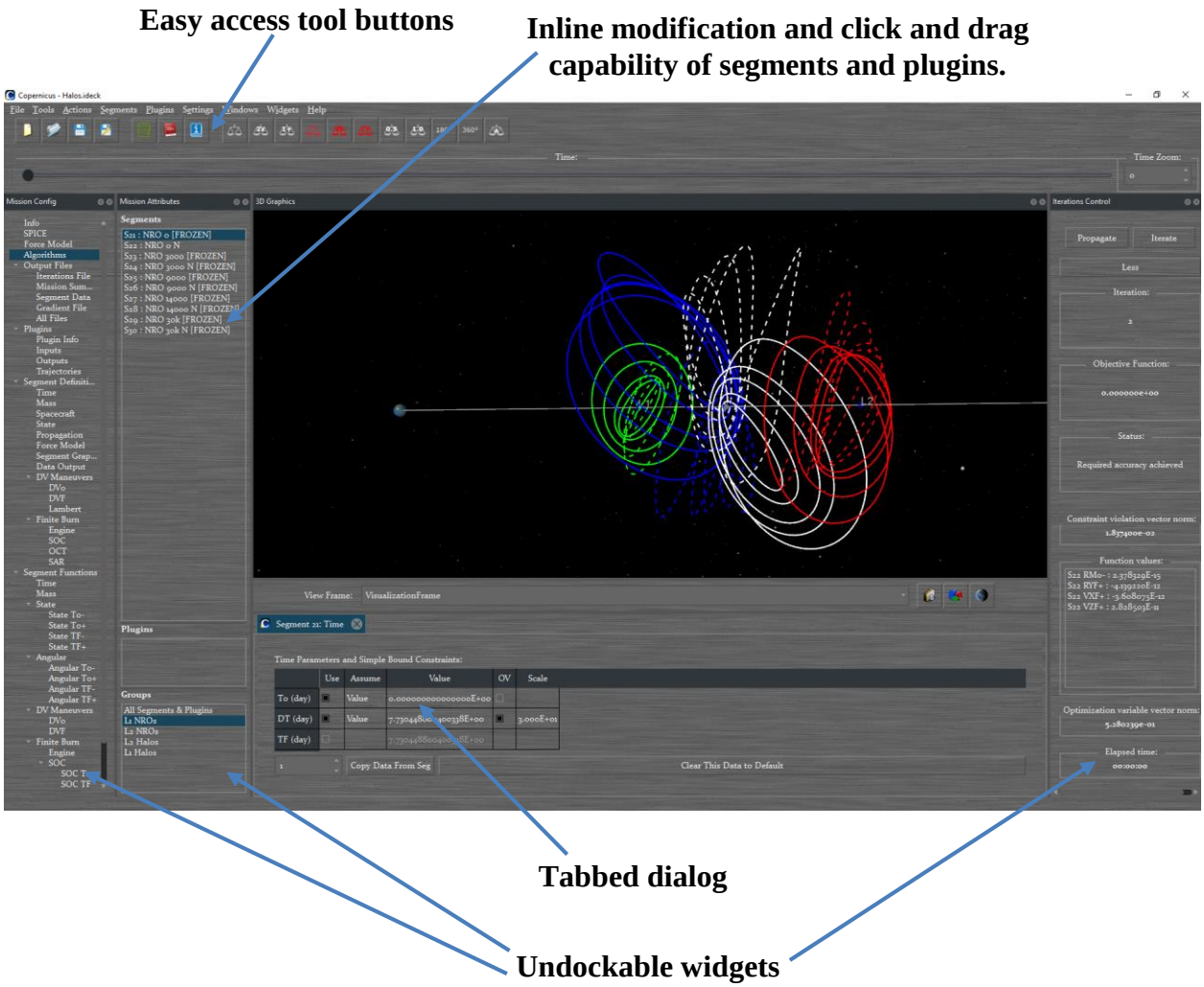



***Figure 6.2-3. GUI main window shown in brushed metal theme in Constantia, size 8 font.***


***Figure 6.2-4. Example dialog (Graphics Options) shown in high visibility/contrast dark button theme, Calibri size 8 font.***

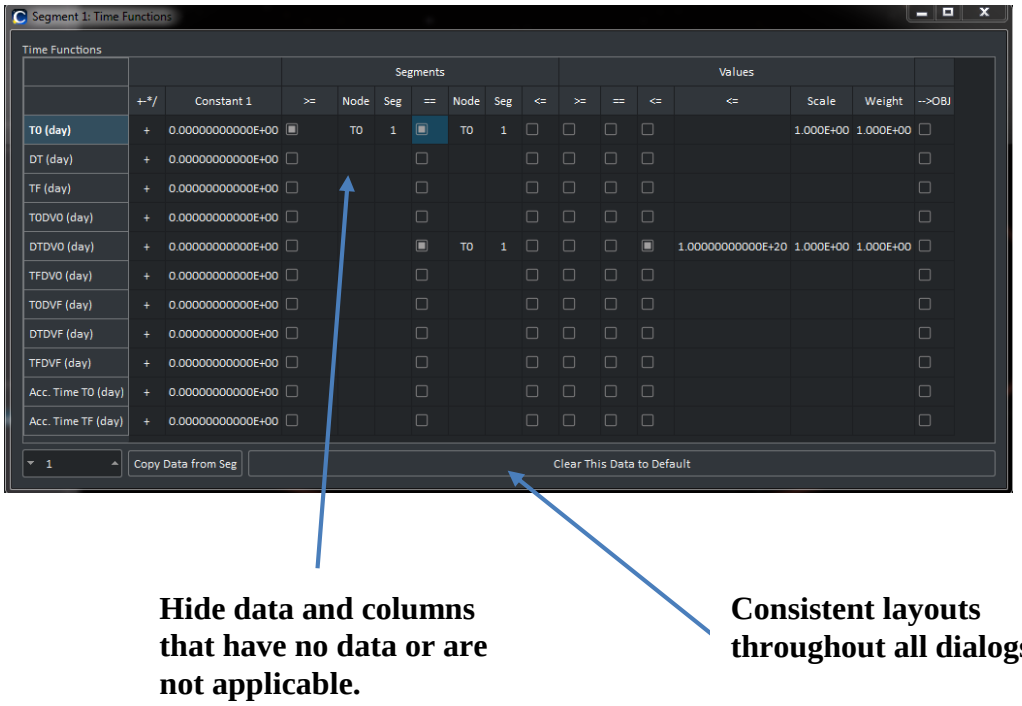



***Figure 6.2-5. Grids support indefinite number of undos/redos to allow users to correct common mistakes quickly.***

## 6.3 3D Graphics Upgrades

Copernicus allows users to perform mission design using a variety of cutting-edge algorithms and design tools. Among these is its 3D interactive visualization interface, which allows users to view the design space from various viewpoints and adjust visualization parameters. Copernicus' 3D visualizations are provided by the OSS OpenFrames API, which allows simulation developers to add real-time 3D interactive visualizations without having to write complex 3D graphics (e.g., OpenGL) code.

The 3D graphics work for this task had two primary goals: (1) improve the OpenFrames API with various features and capabilities needed by the Copernicus code, and (2) the development, integration, and testing of OpenFrames into Copernicus' PyQt-based GUI system. Specific objectives to accomplish these goals were:

- Demonstrate the use of OpenFrames in a PyQt GUI framework
- Demonstrate 3D object interaction in an OpenFrames scene
- Test OpenFrames in a VR environment, and demonstrate how Copernicus can integrate this capability
- Implement planet models of higher fidelity than existing spherical models
- Add light sources and materials for more accurate day/night lighting
- Enhance OpenFrames with additional user-definable viewpoints
- Support Copernicus testing on MacOS and Linux, using the PyQt GUI
- General OpenFrames capability and performance enhancements

### 6.3.1 OpenFrames with Qt and PyQt

Two standalone OpenFrames demonstrations were developed that illustrate the use of OpenFrames to display a 3D scene in a Qt and PyQt GUI. These demonstrations show how to embed OpenFrames in a Qt/PyQt-created window, handle the appropriate callbacks that manage double buffered rendering, and call OpenFrames functions from Python using the developed OpenFrames/ Simplified Wrapper and Interface Generator (SWIG) interface. The OpenFrames/PyQt demonstration was updated to display two OpenFrames scenes in two windows, which enables Copernicus to implement multi-window visualizations.

The next step was adapting this capability into the Copernicus PyQt GUI. One challenge was moving OpenFrames graphics between windows (e.g., from an embedded into a standalone window). This is a necessary use-case because components of the new Copernicus PyQt GUI can be docked within the primary GUI window, or undocked into a standalone window. This challenge was overcome by restructuring how OpenFrames visualizations are paused and resumed to deal with the graphics context moving between windows.

### 6.3.2 3D Object Interaction

A future capability of the Copernicus code is to allow user interaction directly inside the 3D scene. For example, users could obtain information about spacecraft and trajectories by clicking on them with the mouse. An important first step was taken towards this capability by demonstrating object interaction in a standalone OpenFrames demonstration. As shown in Figure .3.2-1, a prototype feature was developed in OpenFrames that allows multiple GUI widgets to be added to the 3D scene. Users can interact with these widgets as they would on a desktop application, and the application can take appropriate steps based on user actions. The Qt GUI toolkit was used for this work, which provides a broad range of standardized widgets (e.g., buttons, sliders, checkboxes, etc.).

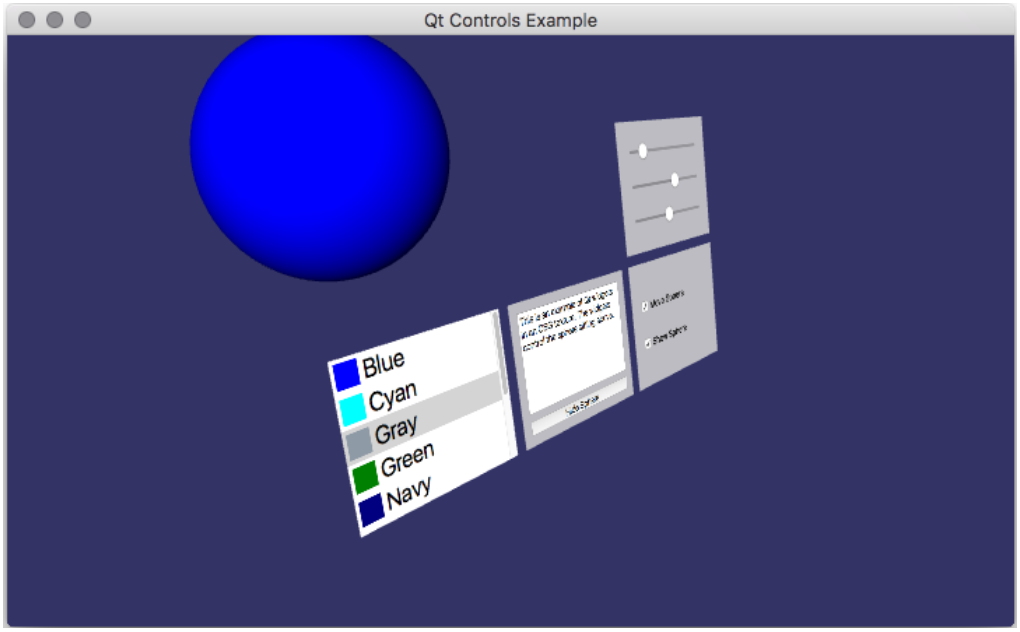



***Figure 6.3.2-1. Interactive widgets in a 3D scene with OpenFrames. This capability will enable full-featured in-scene GUI interactivity in future versions of Copernicus.***

#### 6.3.2.1 OpenFrames in VR

At the start of this task, OpenFrames had the ability to display 3D visualizations in VR, but there were several limitations and issues that prevented the capability from being seamlessly integrated. For example, in certain situations a scene would appear "cut off" near the viewer. Under this task, all outstanding issues were fixed in the OpenFrames VR framework, and significant testing was performed on a Windows 10 system, which is the only operating system that supports VR at this time.

During testing, test cases were first created that used large amounts of data, such as years-long trajectories or point clouds with billions of points, both of which stress even the most powerful computing systems. These test cases were then run on the Oculus Rift and HTC Vive VR headsets (the two main consumer-level VR headsets available at the time), using computers with a range of VR-capable graphics cards (Nvidia GTX 960, 980, and 1080). It was shown that OpenFrames can display large numbers of trajectories, as well as trajectories with many points, with a seamless end-user experience. This VR capability of OpenFrames was then incorporated into Copernicus, and is currently enabled using a command-line flag.

#### 6.3.2.2 Increased-Fidelity Planet Models

Prior to this task, OpenFrames support for rendering planets was limited to spheres. In this task, OpenFrames was enhanced to support two additional forms of planet models:

- Ellipsoids: The OpenFrames sphere can be scaled per-axis to produce a triaxial ellipsoid. This includes the rendering of a latitude-longitude grid overlay using triaxial geodetic ellipsoid parameters. This is primarily useful for highly-ellipsoidal celestial bodies such as some asteroids and moons.
- The osgEarth (a C++ geospatial software developer's kit and terrain engine) API can be used to display a hyper-realistic Earth model (Figure 6.3.2.2-1), using dynamically-accessed online resources for terrain (e.g., Shuttle Radar Topography Mission (SRTM)) and textures (e.g., LandSat or Blue Marble). For example, if given SRTM data, then osgEarth is accurate to 90 m. It can even be given custom data with meter (or less) accuracy and render that just as seamlessly. osgEarth is an open source virtual globe API that is license-compatible with OpenFrames and allows users to configure a dynamic Earth model using extensible markup language (XML) input files. osgEarth can also be used to render models of other celestial bodies (e.g., the Moon or Mars) given the appropriate terrain and texture data, and this capability can be incorporated into Copernicus in future development.

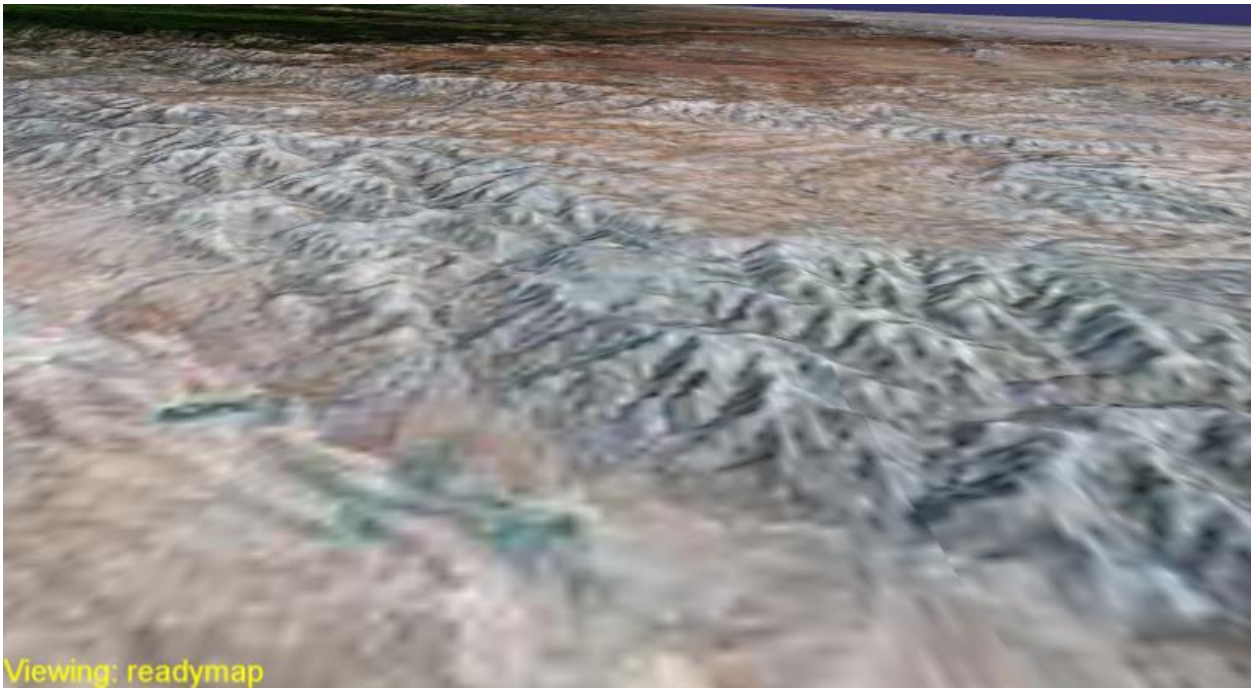



***Figure 6.3.2.2-1. An osgEarth-generated globe in OpenFrames. Hyper-realistic terrain and globe parameters enable Copernicus to properly visualize all mission phases and enhance its usability and appeal as a mission design tool at NASA.***

### 6.3.2.3 Lighting and Material Support

Support for light sources was added (e.g., the Sun) to OpenFrames. Common light properties can be customized (e.g., location, and ambient, diffuse, and specular components). Additionally, customizable materials were added to all sphere objects. This enables specification of reflectivity and emission components for celestial bodies, and separate day and night textures. When sunlight is combined with correct materials, the resulting visualizations can be realistic and most importantly useful to determining lighting conditions for spacecraft and trajectories (see Figure 6.3.2.2-2).

Currently, the lighting feature can be used only for visual validation of expected lighting conditions (e.g. to check if the correct side of the spacecraft is lit based on its current orientation). This is a necessary first step towards more advanced lighting and engineering use-cases, including:

- Extract the light intensity shining on the rendered spacecraft and translate that to a power estimate using knowledge of solar panel efficiency
- Implement shadowing, which enables knowledge of when spacecraft parts are blocking light from reaching solar panels or instrument cameras

These advanced features are not currently available, but the work in this task has laid the groundwork for capabilities like these to be added as future work.

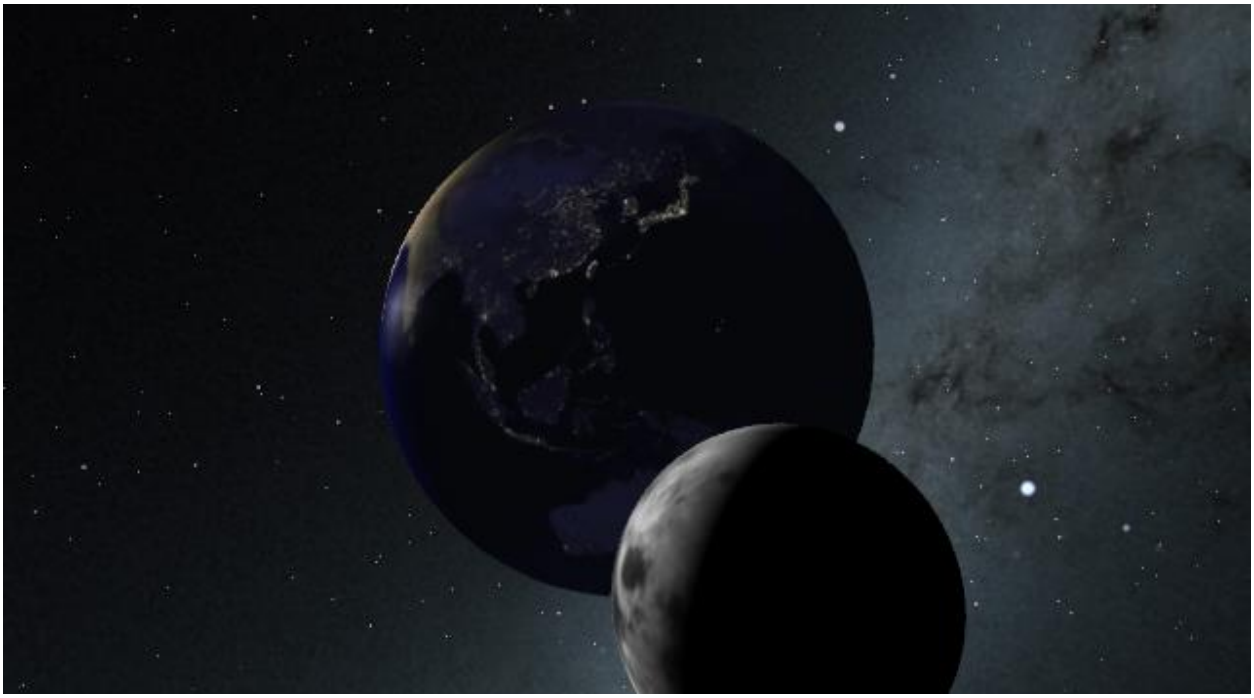


***Figure 6.3.2.2-2. Accurate Sun-based lighting on the Earth and Moon. Apart from its visual appeal, accurate lighting enables Copernicus to validate day/night visibility conditions for spacecraft.***

### 6.3.2.4 Enhanced Viewpoints

OpenFrames can be used to define multiple cameras that follow various objects in the scene (Figure 6.3.2.4-1). The ability was added for cameras to track secondary objects while following the primary object. This allows users to follow an object while always maintaining their view towards a secondary object of interest. For example, this is useful in cluster flight applications or to determine eclipsing of an Earth-orbiting satellite.

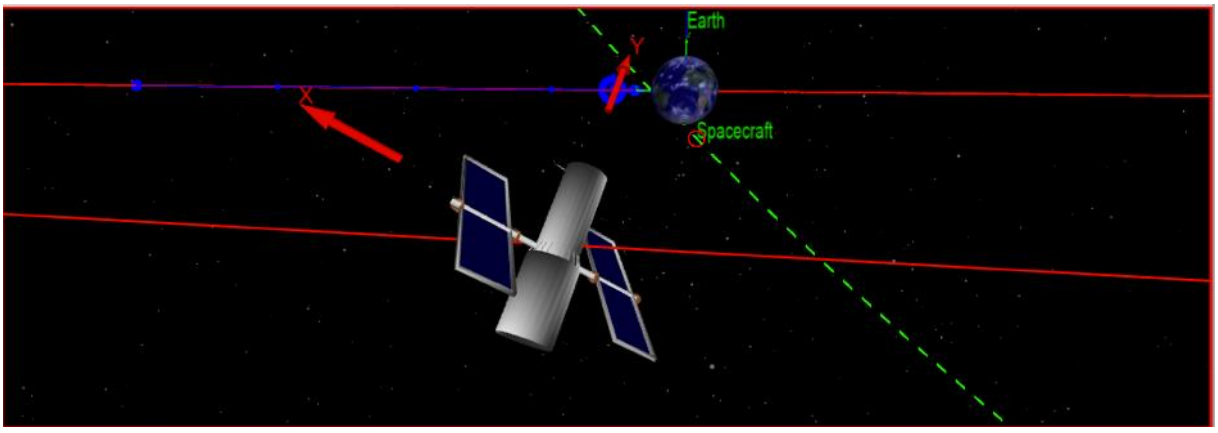



***Figure 6.3.2.4-1. Viewing a secondary spacecraft as seen from the primary spacecraft. To-From views like this make it possible to visually identify times when line-of-sight communications between spacecraft are lost.***

#### 6.3.2.5 Benefits to NASA Programs

In addition to Copernicus, OpenFrames is used by multiple NASA programs, the GMAT, NASA Goddard Space Flight Center (GSFC), and multiple applications created by the GSFC VR Working Group. Since OpenFrames is OSS, its core usage philosophy is that advancements provided by a project should provide a benefit to other NASA programs and projects. Because of this philosophy, improvements made in this assessment to OpenFrames have been contributed to the public OpenFrames code repository.

One OpenFrames benefit is GMAT has incorporated the "Lighting and Material Support" and "Enhanced Viewpoints" updates. Because of these updates, GMAT's visualizations show accurate daytime and nighttime conditions for celestial bodies and spacecraft at any given time. These updates enable users to define custom camera views that look towards bodies of interest (e.g., from the Lunar Reconnaissance Orbiter (LRO) to the Moon). This has proven to be a powerful capability in GMAT, and has increased its desirability for real-time operations in GSFC Mission Operations Centers.

Another example of the multi-program benefit of this assessment comes from the GSFC Virtual Landscapes (VL) project, which shows high-density planetary Light Detection and Ranging (Lidar) data to scientists for virtual exploration. VL overlays the Lidar data on a hyper-accurate Earth model provided by osgEarth. This capability was made possible by the "Increased-Fidelity Planet Models" updates to OpenFrames.

### 6.3.3 Synergy with Other NASA Programs

Emergent Space Technologies has performed considerable research and development on OpenFrames under NASA Phase II SBIR (i.e., NNX16CG16C). Based on the enhancements provided by this assessment, the NASA SBIR has enabled a Phase II-X extension of this effort which will develop additional technologies of interest to Copernicus:

- Development of novel user interfaces inside a 3D scene. This is an extension of the "3D Object Interaction" task, and will enable advanced Qt-based UI widgets inside the 3D scene. This will further enhance the ability to obtain information about spacecraft and trajectories.
- Ability to render accurate shadows of celestial objects on each other and on spacecraft. Computing shadowing is an advanced component of space mission design, and visualizing these shadows in Copernicus will allow our users to more quickly determine shadow-based constraints and adjust their mission designs accordingly.

This is an ideal example of multiple organizations (e.g., NESC and SBIR) collaborating to provide a significant boost in capability to OpenFrames and all projects that use it, including Copernicus.

Another example of multi-organization synergy comes from the VL project. VL required 3D UI widgets inside the VR scene, which is an extension of the "3D Object Interaction" task. VL supported a Phase III SBIR [ref. 7] to develop and test the prototype developed under this task using VR hardware including Oculus Rift and HTC Vive. These updates were committed to the OpenFrames code repository, and are available for Copernicus when interactive UI widget and VR support are implemented. As before, the benefits of this Phase III SBIR were made possible by this assessment's development of a 3D UI prototype, but were supported by the VL project.

### 6.3.4 MacOS and Linux Testing and Support

The PyQt-based GUI allows Copernicus to support MacOS and Linux with a modern look-and-feel, and OpenFrames was updated to support these operating systems. Some notable updates include:

- Support for Retina displays, which are standard on MacBook laptops. Retina displays have high pixel densities, and Qt handles this by defining a "virtual pixel" that is different from a physical pixel. This caused OpenFrames to render to only a portion of the desired 3D window. It was found that there is a ratio between virtual and real pixel size that can be retrieved from PyQt. Applying this ratio to Copernicus allows full support for Mac Retina displays.
- MacOS handles 3D graphics contexts differently than Windows or Linux, and as a result resizing the 3D graphics window resulted in graphical issues. It was found that Copernicus was not using an OpenFrames function that resets the graphics context when the window is resized, which is required on MacOS. The ability to resize a window was restored on MacOS after applying this update.
- To run Copernicus on MacOS or Linux, users were required to set environment variables to appropriate values so that the necessary third-party libraries (e.g., OpenFrames and OpenSceneGraph) could be found by Copernicus. The ability was developed to encode this information directly into the Copernicus application during build-time, so that the application can find the prerequisite libraries without end-user input.
- A Copernicus use-case is running the application remotely on a Linux server, with visualizations being displayed on the local machine. This requires the ability to forward the visualizations generated on the server to the user's machine. OpenFrames was updated to support this use-case, and performed testing on a NASA remote Linux server. Formerly, if a large scan was run on a Linux server, the user would have to download the resultant Copernicus mission files to their local machine to open them. Now this can all be done on the server. It also means that Copernicus can be used from machines that do not have Copernicus installed by accessing the server remotely.

### 6.3.5 Additional OpenFrames Enhancements

In addition to the primary enhancements discussed, the following improvements to OpenFrames' capabilities and performance were implemented:

- Updated OpenFrames to the most recent build of OpenSceneGraph 3.6.3 (a core dependency). This contains feature improvements (e.g., text rendering, better multi-core support, and full OpenGL Core Profile support for MacOS).
- Support for custom fonts and sizes.
- Support for heads-up-display (HUD) style text overlaid on main 3D scene.
- Antialiasing support, in standalone OpenFrames windows and in Qt and PyQt-generated windows. This feature eliminates jagged line edges and results in a more professional 3D visualization. It was fundamentally unsupported under the Winteracter GUI, and is one of many significant Qt GUI improvements.
- Overhauled the OpenFrames time management system. It is simpler to specify the simulation time and rate during animation with more accurate synchronization with the wall clock.

- Significant performance enhancement when adding the state and time data points to a 3D trajectory (e.g., after an integration step). Previously, if Copernicus attempted to add a point while the OpenFrames rendering thread was processing the trajectory, then Copernicus would be forced to wait until the processing was completed. With updates, Copernicus no longer needs to wait on the OpenFrames thread when adding new points, which has resulted in a significant speedup when adding points to a trajectory.
- Fixed a thread starvation condition when attempting to pause animation. Thread starvation occurs when multiple computing threads attempt to access the same resource, but one of them is prevented from doing so for a long time (i.e., is "starved"). In this case, Copernicus would request OpenFrames to pause animation when loading a new input deck, but the OpenFrames rendering thread would prevent the request from being completed. This resulted in Copernicus waiting for OpenFrames to pause animation, and end-users observed this as Copernicus "freezing" while opening an input deck. This issue was eliminated by placing higher priority on pause requests than on rendering, so that the request is processed instantly. This resulted in a significant speedup when loading input decks.

### 6.4 Cross-Platform Capability – Linux and Mac Versions

Prior to this task, the full version of Copernicus with the GUI and 3D graphics only existed on the Windows platform (i.e., version 4.4). A command-line only version existed for Linux, and no version existed for MacOS. Winteracter was fully functional on Windows, and to a limited extent on MacOS and Linux where it was built on the outdated OpenMotif GUI toolkit. One of the inherent benefits of the Qt-based GUI is that the same source code can be built for multiple platforms. This reduces the amount of source code and facilitates maintainability and full cross-platform development. In addition, the old GUI required the user to run Copernicus on a virtual machine application when using a Mac. For the Mac version, the new GUI implementation eliminates the slow refresh rates of the original Mac GUI and the unreliable operation of Copernicus on a virtual machine.

With the use of the PyQt toolkit, a fully cross-platform GUI is available. The GUI is fully functional and has the same behavior across Windows, MacOS and Linux platforms. This capability allows users to run Copernicus natively on a preferred computer platform. See Figure 6.4-1 for a screenshot of Copernicus running on Linux. Various changes were made to the code to enable this capability.

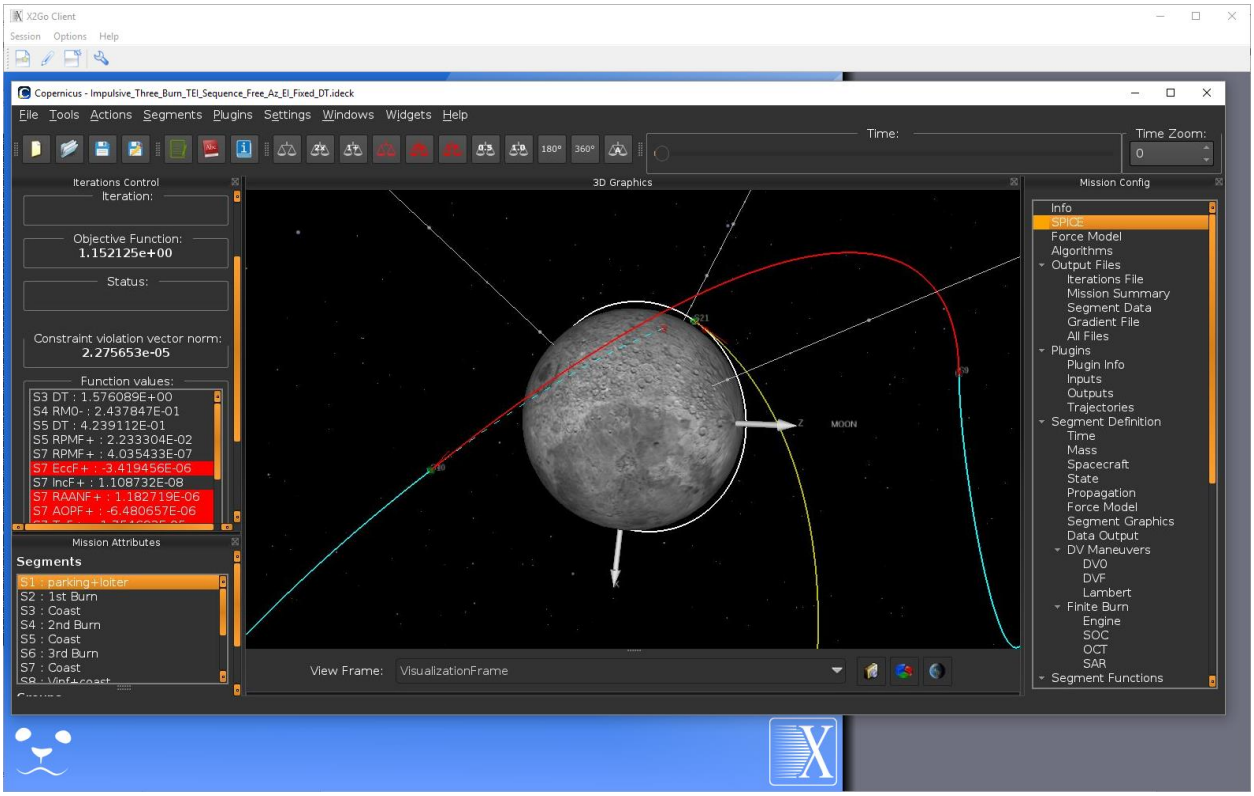



***Figure 6.4-1. Copernicus running on Linux via the X2Go remote desktop client. The look and behavior of the tool is identical on Windows, Linux, and MacOS platforms.***

### 6.4.1 Cross Platform Make (CMake)

Additionally, a CMake build system was created to take advantage of its benefits. CMake is a modern open-source, cross-platform software build system that can be used to dynamically generate the files needed to build Copernicus for a given platform (e.g., Visual Studio, Make, Eclipse, etc.). CMake automatically discovers system libraries and toolchains across platforms. It dynamically generates platform specific build systems that do not have to be under version control with the source code. This makes portability and initial setup of the build system on any supported platform easier with no need to maintain platform or build system specific files. Previously, there were two sets of files that were needed (i.e., Visual Studio files for Windows, and Make files for Linux), and each set of files had to be under revision control and maintained separately with redundant information. CMake reduces these files to a set of text files that contain all the information to generate the build system files. The result is a greatly-simplified cross-platform development process.

### 6.4.2 Software Development Practices

This assessment allowed for Copernicus to upgrade to the Git version-control system. Previously, the Subversion (SVN) version-control system was used. For the Copernicus development tasks, it was found that Git provided a number of desired development capabilities over SVN, including better branching and merging features which are useful for simultaneous development. In addition, a web-based JSC GitLab server is now being used for Copernicus centralized control and issue tracking of tickets.

### 6.4.3 Copernicus as a Service

Previous builds of Copernicus were monolithic applications that lacked a user-callable API. The limitations of this approach became apparent and it was decided to move the tool toward a new mode where it could be called by other tools as a service. To facilitate this, Copernicus was rearchitected into a shared library: a Dynamic Link Library (DLL) on Windows, a Dynamic Library (DYLIB) on MacOS, and a Shared Object (SO) file on Linux. This library contains all the functionally of Copernicus, which allows the possibility of other systems incorporating this library to enable access to this functionally. Currently, this mode has been incorporated into the PyQt GUI. A forward work item is to expand the API to allow for more access to the core features of Copernicus (e.g., as part of the new Python scripting environment). This approach will provide a powerful toolset for a variety of applications (i.e., allowing a user-created script access to the force models, integrators, and ephemerides that are already built-into Copernicus).

### 6.4.4 Improved Plugins Capabilities

Copernicus, while a capable tool, focuses on numerical trajectory optimization. The plugin technology allows Copernicus to employ an unlimited number of possible externally-developed algorithms. For example, one plugin could provide a closed-loop guided trajectory in place of a Copernicus segment. Copernicus does not inherently contain guidance algorithms, but the plugin capability allows the user to incorporate a guidance algorithm to assess how the spacecraft might fly with the onboard flight software in a 3-degree-of-freedom mode. See Figure 6.4.4-1 for a sample schematic of a Copernicus mission using plugins. Other possible examples could range from an orbit monitoring trajectory that inserted orbit maintenance burns when needed, complex Earth entry interface target lines, mass versus delta-velocity (ΔV) equations or algorithms for assessing spacecraft size with ΔV requirement for preliminary vehicle sizing.

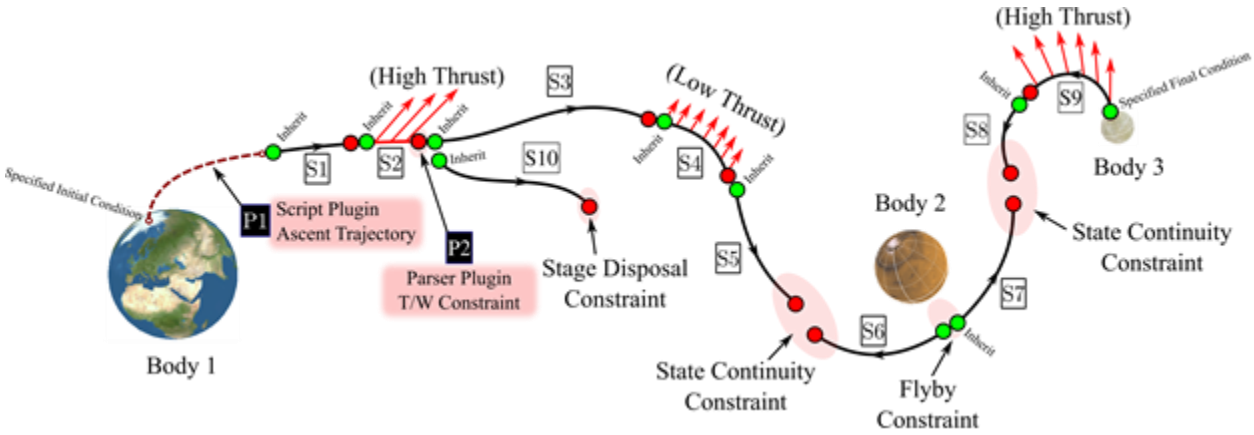



***Figure 6.4.4-1. A Copernicus mission is constructed from segments and plugins. Segment are built-in components, whereas plugins are used to incorporate user-defined algorithms.***

A feature was added to Copernicus to specify custom coordinate frames via DLL plugins. This provides a mechanism for Copernicus to use a user-defined frame. Various other new frame-related features were added in the version 4.6 to make it easier to use frames for various purposes, and to enable increased flexibility in Copernicus to match other tool frames. This feature is being used by the Orion MPCV Program, to match the frames used in Copernicus with the MPCV flight software and other analysis and operations tools.

During MPCV mission analysis, it was discovered that a significant source of error between modeling by various engineering teams was the result of inconsistent usage of Earth orientation parameters. In an effort to unify trajectory modeling, Copernicus needed a way to provide custom, dynamic Earth orientation modeling beyond what could be provided by Jet Propulsion Laboratory's (JPL) Spacecraft, Planet, Instrument, C-Matrix & Events (SPICE) library. A frame plugin, utilizing the new interface developed during this task, was developed to provide an Earth-fixed frame via the International Astronomical Union's (IAU) Standards of Fundamental Astronomy (SOFA) software library. The plugin parses a JavaScript Object Notation (JSON) text configuration file that specifies the parameters to use when computing Earth orientation, providing finer control than previously available. Currently, the plugin provides the ability to use single constant values for the parameters. A planned enhancement will allow a table of time-varying parameters to be provided (e.g., directly from files provided by the International Earth Rotation and Reference Frames Service (IERS)).

Various other plugin-related improvements were made:

- DLL plugins can access the SPICE environment within Copernicus. This allows for plugins to access data from the SPICE pool (e.g., the ephemeris, reference frames and gravitational parameters). This feature allows for greater interaction between user-created plugins and Copernicus.
- A plugin support library was also created that can be used by users when creating DLL plugins. This library provides various modules that are useful for building DLL plugins and interfacing with Copernicus. This library makes it easier for users to create plugins, and will be expanded as future capabilities are added.
- Added plugins to the "groups" feature in Copernicus, which can be used to define multiple optimization problems in the same mission file. These groups can contain plugins. This updated "finishes" the groups feature, and provides Copernicus users with maximum flexibility to define different optimization problems in a mission. Groups are being used for the MPCV EM-1 mission design.

### 6.4.5 Software Architecture Improvements

This assessment provided an opportunity for upgrades to the Copernicus software architecture. Copernicus takes advantage of the strengths of three programming languages (see Figure 6.4.5-1): Fortran 2008 for the mathematical core of the program, Python for the GUI and scripting, and C++ for 3D graphics and OpenGL interfacing.

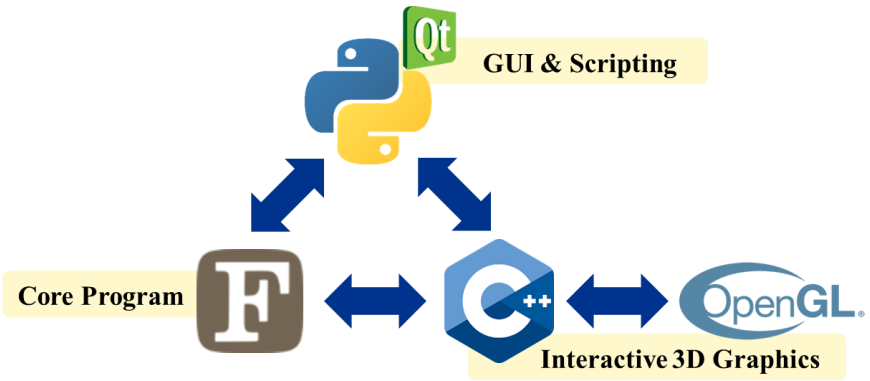



***Figure 6.4.5-1. Copernicus takes advantage of the strengths of Fortran 2008, Python, and C++.***

Numerous software architecture updates were made as part of this task. Several of the significant changes were:

- Changes were made to the method that segment data is exported to the files. Output files are cached internally and generated after the segment is propagated. The output files are generated faster. For example, on one of the development machines, a 30-day propagation with data file generation of a low-Earth orbit using an 8x8 gravity model, using DDEABM (i.e., Adams-Bashforth-Moulton variable step-size integrator) with a tolerance of 1e-12 took about 9 seconds with version 4.4, now takes about 4 seconds with version 5.0.
- A new option was added to export the segment data to Hierarchical Data Format (HDF5) binary format. This is a standard format used for scientific data, and makes it easy to read the Copernicus-generated data by other tools that support this format.
- New user-friendly tools were added for finding eclipses, performing time transformations, and performing state transformations.
- A new ramp control law option was added for finite burn engines. This control law uses cubic spline equations to transition from the initial and final values to the ramp phase. This feature is being used for SLS/MPCV to model the SLS startup and shutdown transients.
- Added tri-axial ellipsoid state parameters were incorporated, which are useful for ellipsoidal-shaped solar system bodies (e.g., asteroids).
- A new feature was added to allow the option of specifying SPICE pool variables in JSON files. This is an alternative to the native text Planetary Constants Kernel (PCK) file format supported by SPICE. The file format is a list of the pool variables and their values. One advantage of Copernicus JSON kernels is they are cross-platform. Normal SPICE PCK files are not cross-platform, which requires separate Windows and Mac/Linux versions. Another advantage is that they can be easily created, parsed, and manipulated by other tools.
- Various new command line options were added, improving the flexibility of using Copernicus from the command line.

Numerous other features and improvements were made. See the version 4.5 and 4.6 Copernicus release memos in Appendices A and B, respectively for details of these changes.

### 6.4.6 Enhanced Python Scripting Capabilities

The Copernicus capabilities are enhanced when matched with the ability to automate execution and to solve problems in large batches. To that end, a package called CopPy was developed, which provided a Python interface to Copernicus. However, CopPy only allowed modification of existing items in a Copernicus mission file (i.e., input deck). CopPy could not be used to construct missions (or individual trajectory segments from scratch, or add new elements (e.g., finite burns) that were not in the original file. The package provided access to a limited subset of input deck configuration values and actions. While a variety of workarounds to these limitations were developed during practical use, a more robust and complete interface providing all Copernicus options was desired. RoboCopPy, which was a new Python package for interfacing with Copernicus, was developed to satisfy this need.

The new package is an object-oriented Python approach to the problem. GUI fields have been organized into classes that correspond to familiar dialogs. The classes provide a complete mapping of every option and field available in each GUI dialog. This approach makes developing scripts easier. By nature of its object-oriented design, RoboCopPy allows construction of complete input decks by adding class instances to various collections. Figure 6.4.6-1 is a simplified example of code to create an input deck in Python that represents a prototypical International Space Station (ISS) trajectory.

```
import robocoppy as rcpy

iss = rcpy.Ideck()

seg = rcpy.Segment("ISS")

seg.mass.m0mm.value = 400000
# For "generic" quantities, you don't need to specify the `.value`.
seg.time.t0 = 0

j2k = rcpy.Frame()
j2k.frametype_id = rcpy.CopFrameEnum.j2000
j2k.framecenter_id = rcpy.FrameCenter.main
j2k.mainbody = rcpy.SpiceBodyEnum.earth

s = rcpy.State()
s.frame = j2k

p = s.param
p.params_id = [rcpy.Param1Enum.sma, rcpy.Param2Enum.ecc,
               rcpy.Param3Enum.inc, rcpy.Param4Enum.raan,
               rcpy.Param5Enum.aop, rcpy.Param6Enum.ta]
p.angle_unit = rcpy.AngleUnits.deg

s.sma = 6778.0
s.ecc = 0.001
s.inc = 51.66
s.raan = 276.838
s.aop = 69.6
s.ta = 189.3489

seg.state = s
iss.segments = [seg]
iss.save('iss.ideck')
```

***Figure 6.4.6-1. Simplified example of code to create an input deck in Python that represents a prototypical International Space Station (ISS) trajectory.***

A useful feature shown in Figure 6.4.6-1 is the ability to refer to state elements by user-friendly names (e.g., “sma”), once the state parameterization has been configured. State elements may be referenced by index (e.g., `s.state[0].value` for the first state element).

Among the most powerful features of RoboCopPy is the implementation of segment references. Throughout an input deck, values can be inherited between segments or constrained by references to other segments. When loading an input deck into RoboCopPy, these references are converted to Python object references. These references are converted to their Copernicus

segment numbers when the final input deck is saved. This allows for reordering of segments without needing to manually update numeric references throughout the file. Segment references can optionally be made using segment names which are automatically resolved to the appropriate Python object reference. Assuming all segment names in the input deck are unique, this provides a useful way to refer to segments without needing to know their location in the sequence. Because a RoboCopPy input deck is a collection of segment object references, multiple input decks can refer to the same segment object in memory, and allowing for easy reuse. An example demonstrating these references is shown in Figure 6.4.6-2.

```
>>> from robocoppy import Ideck, Segment
>>> seg_a = Segment('Segment A')
>>> seg_b = Segment('Seg B')
>>> seg_c = Segment('Seg C')
>>> seg_d = Segment('Seg D')
>>> seg_b.time.t0.inherit_seg = seg_a  # not ambiguous!
>>> seg_d.mass.m0mm.inherit_seg = 1  # this is ambiguous...
>>> x = Ideck()
>>> x.segments = [seg_a, seg_d, seg_b]
>>> # now this will refer explicitly to seg_a
>>> seg_d.mass.m0mm.inherit_seg.parent = x.segments
>>> seg_d.mass.m0mm.inherit_seg.segment.name
'Segment A'
>>> y = Ideck()
>>> y.segments = [seg_c, seg_a, seg_d]
>>> seg_c.sc_data.dry_mass.inherit_seg = 'Segment A'  # this is ambiguous
>>> y.save('foo.ideck')  # now this refers explicitly to seg_a
>>> seg_c.sc_data.dry_mass.inherit_seg.segment.name
'Segment A'
```

***Figure. 6.4.6-2. Example of RoboCopPy input deck segment references.***

In cases where a new segment may be spliced into an existing trajectory, it may be necessary to update segment references in batch from one segment to another. Several methods exist to facilitate this action. The Segment class provides methods **`replace_inherits_with_seg`**, to replace all references within that segment from one to another, or **`replace_list_inherits_with_me`**, which replaces all references in a provided list to a given segment with a reference to the current one.

These features, among other enhancements, make RoboCopPy a superior method for creating, manipulating, and running Copernicus input decks in scripts. The RoboCopPy beta version has facilitated the development of complex MPCV EM-1 abort trajectory studies with scripts to generically model various scenarios. The originally planned effort to generate the complete set of scenarios for the study would have taken many months to write scripts full of workarounds to the limitations of CopPy. The advantages of RoboCopPy reduced the time to complete and the team size required to develop the scenarios by nearly half.

It is envisioned the RoboCopPy Python interface will be merged with the Python (PyQt) GUI. This could provide significant new capabilities (e.g., real-time manipulation of the mission by the user from within the GUI). This could be enabled by a set of built-in or user-defined macros, which are Python scripts that operate on the current mission and update the GUI accordingly. This could make it easier to perform repetitive tasks, and adding new capabilities to the tool via Python-based GUI plugins.

### 6.4.7 Bug Fixes

As part of the development activity, numerous bugs were fixed. For details on these fixes, see the 4.5 and 4.6 release memos in Appendices A and B. As part of this task, the version 5.0 was also beta tested by users at various NASA Centers.

## 7.0 Summary

The purpose of this assessment was to develop updates and new features for the NASA Copernicus Trajectory Design and Optimization tool for application to NASA programs and projects. These updates significantly improve the ability to design and optimize complex trajectories over multiple trajectory phases and allow the use of unique vehicle-specific guidance, control, and trajectory strategies and constraints, and the creation of an almost unlimited number of unique user-defined capabilities. Products of the assessment are major upgrades to the Copernicus tool that provide significant enhancements and effectiveness to the user. The plug-in technology will open Copernicus to compatibility with an unlimited possibility of user-defined algorithms that bi-directionally interact with the tool. The platform availability of Copernicus has been expanded from PC-only to include Mac- and Linux-based operating systems.

Products of this assessment will likely be used by Agency organizations, NASA-associated contractors, and academia to provide potential benefit to all analysts, project leads, and managers involved in trajectory design and optimization. The assessment currently affects multi-Center projects and programs (e.g., Orion MPCV, SLS, and the Lunar Orbital Platform Gateway). Additionally, it affects numerous government, commercial, and/or academic programs, proposals, and studies requiring trajectory design and optimization.

To obtain the latest version of Copernicus, and the user guide, go to: https://software.nasa.gov/software/MSC-25863-1 and click on "Request Now". The updated Copernicus User Guide is also accessible via hotlinks within the Copernicus GUI.

## 8.0 Findings, Observations, and NESC Recommendations

The following findings were identified:

### 8.1 Findings

**F-1.** The PyQt GUI provides significant new capability and increased usability of the Copernicus tool.

**F-2.** The 3D graphics improvements (e.g., OSS OpenFrames) provide increased capabilities to Copernicus. These same upgrades can be applied to other NASA tools (e.g., a current GMAT development activity includes integration of OpenFrames graphics capability).

**F-3.** The new RoboCopPy Python scripting interface provides significant new capability for scripting Copernicus.

**F-4.** Plugin improvements increase the ability of Copernicus to interoperate with other tools and provided an ability to match external models used in analysis and operations.

**F-5.** The use of standard data formats (e.g., HDF5 and JSON) greatly improves the ability of Copernicus to interoperate with other NASA analysis tools.

### 8.2 Observations

The following observations were identified:

**O-1.** Copernicus is used in many projects across the Agency, including being the primary trajectory optimization tool used in the integrated SLS/MPCV missions for pre-mission design and has been identified as one of the software tools to be used in mission operations.

**O-2.** The beta release of Copernicus 5.0 has facilitated the development of complex MPCV EM-1 abort trajectories and reduced the time to complete and the team size required to develop the scenarios by nearly half.

**O-3.** NASA Glenn Research Center (GRC) users have tested the beta release of the Copernicus GUI and found that it provides a fast and reliable trajectory modeling tool that runs on Macs, eliminating the slow refresh rates of the original Mac GUI and the unreliable operation of Copernicus on Virtual Machine. This has resulted in a reduction in the time required to develop or modify trajectory models in Copernicus and has greatly increased usability. The beta version is being used for trajectory analysis supporting NASA GRC's Compass Team and the Mars Study Capability Team.

**O-4.** There are continued demands for enhanced existing capability and adding new capabilities in Copernicus.

### 8.3 NESC Recommendations

The following NESC recommendations were identified and directed towards Copernicus users:

**R-1.** Upgrade to Copernicus version 5.0 to take advantage of new features and enhancements. (***F-1 through F-5)***

## 10.0 Other Deliverables

No unique hardware, software, or data packages, outside those contained in this report, were disseminated to other parties outside this assessment

## 12.0 Recommendations for NASA Standards and Specifications

No recommendations for NASA standards and specifications were identified as a result of this assessment.

## 13.0 Definition of Terms

| | |
|---|---|
| Corrective Actions | Changes to design processes, work instructions, workmanship practices, training, inspections, tests, procedures, specifications, drawings, tools, equipment, facilities, resources, or material that result in preventing, minimizing, or limiting the potential for recurrence of a problem. |
| Finding | A relevant factual conclusion and/or issue that is within the assessment scope and that the team has rigorously based on data from their independent analyses, tests, inspections, and/or reviews of technical documentation. |

| | |
|---|---|
| Lessons Learned | Knowledge, understanding, or conclusive insight gained by experience that may benefit other current or future NASA programs and projects. The experience may be positive, as in a successful test or mission, or negative, as in a mishap or failure. |
| Observation | A noteworthy fact, issue, and/or risk, which may not be directly within the assessment scope, but could generate a separate issue or concern if not addressed. Alternatively, an observation can be a positive acknowledgement of a Center/Program/Project/Organization's operational structure, tools, and/or support provided. |
| Problem | The subject of the independent technical assessment. |
| Proximate Cause | The event(s) that occurred, including any condition(s) that existed immediately before the undesired outcome, directly resulted in its occurrence and, if eliminated or modified, would have prevented the undesired outcome. |
| Recommendation | A proposed measurable stakeholder action directly supported by specific Finding(s) and/or Observation(s) that will correct or mitigate an identified issue or risk. |
| Root Cause | One of multiple factors (events, conditions, or organizational factors) that contributed to or created the proximate cause and subsequent undesired outcome and, if eliminated or modified, would have prevented the undesired outcome. Typically, multiple root causes contribute to an undesired outcome. |
| Supporting Narrative | A paragraph, or section, in an NESC final report that provides the detailed explanation of a succinctly worded finding or observation. For example, the logical deduction that led to a finding or observation; descriptions of assumptions, exceptions, clarifications, and boundary conditions. |

## 14.0 Acronyms and Nomenclature List

| | |
|---|---|
| 3D | Three-Dimensional |
| API | Application Programming Interface |
| CMake | Cross Platform Make |
| COTS | Commercial-Off-The Shelf |
| DDEABM | Adams-Bashforth-Moulton variable step-size integrator |
| DLL | Dynamic Link Library |
| DRO | Distant Retrograde Orbit |
| DYLIB | Dynamic Library (MacOS) |
| EM-1 | Exploration Mission 1 |
| GMAT | General Mission Analysis Tool |
| GRC | Glenn Research Center |
| GSFC | Goddard Space Flight Center |
| GUI | Graphical User Interface |
| HDF5 | Hierarchical Data Format (version 5) |
| HUD | Heads-Up-Display |

| | |
|---|---|
| IAU | International Astronomical Union |
| IERS | International Earth Rotation and Reference Frames Service |
| ISS | International Space Station |
| JPL | Jet Propulsion Laboratory |
| JSC | Johnson Space Center |
| JSON | JavaScript Object Notation |
| LaRC | Langley Research Center |
| LRO | Lunar Reconnaissance Orbiter |
| MPCV | Multi-Purpose Crew Vehicle |
| NASA | National Aeronautics and Space Administration |
| OSS | Open Source Software |
| PCK | Planetary Constants Kernel |
| PyQt | Python Interface to Qt |
| SBIR | Small Business and Innovative Research |
| SLS | Space Launch System |
| SO | Shared Object |
| SOFA | Standards of Fundamental Astronomy |
| SPICE | Spacecraft, Planet, Instrument, C-Matrix & Events |
| SRTM | Shuttle Radar Topography Mission |
| SVN | Subversion |
| SWIG | Simplified Wrapper and Interface Generator |
| UI | User Interface |
| VL | Virtual Landscapes |
| VR | Virtual Reality |
| XML | Extensible Markup Language |

## 15.0 References


1. J. Williams, “Copernicus Version 4.5,” JSC Engineering, Technology and Science (JETS) Contract, Technical Brief JETS-JE23-17-AFGNC-DOC-0066, December 12, 2017.
2. J. Williams, “Copernicus Version 4.6,” JSC Engineering, Technology and Science (JETS) Contract, Technical Brief JETS-JE23-18-AFGNC-DOC-0009, April 20, 2018.
3. C. Ocampo, “An Architecture for a Generalized Trajectory Design and Optimization System,” in Proceedings of the Conference: Libration Point Orbits and Applications (G. Gómez, M. W. Lo, and J. J. Masdemont, eds.), pp. 529–572, World Scientific Publishing Company, June 2003. Aiguablava, Spain.
4. J. Williams, R. D. Falck, and I. B. Beekman. “Application of Modern Fortran to Spacecraft Trajectory Design and Optimization,” 2018 Space Flight Mechanics Meeting, AIAA SciTech Forum, (AIAA 2018-1451).
5. J. Williams, “A New Plugin Architecture for the Copernicus Spacecraft Trajectory Optimization Program,” AAS/AIAA Astrodynamics Specialist Conference, Vail, Colorado, August 2015. AAS 15-606.
6. J. Williams, J. S. Senent, D. E. Lee, “Recent Improvements to the Copernicus Trajectory Design and Optimization System,” Advances in the Astronautical Sciences, 2012.
7. SBIR Contract 80NSSC18P0728. The COR was Thomas Grubb, NASA GSFC, thomas.g.grubb@nasa.gov.

## Appendices

A. Copernicus Release 4.5.0
B. Copernicus Release 4.6.0

# Appendix A. Copernicus Release 4.5.0

**NASA Johnson Space Center**
**EG/Aeroscience and Flight Mechanics Division**
**Flight Dynamics Team Technical Brief**

## Copernicus Version 4.5

FltDyn-CEV-17-66

December 12, 2017


**Prepared by:**

**Jacob Williams**
Senior Astrodynamics Engineer
GN&C and Aerosciences Section – JE23
JETS Engineering Department

**Christopher W. Foster**
Senior GN&C Engineer
Odyssey Space Research, LLC.

**Approved by:**

**Gerald L. Condon**
EG/Aeroscience and Flight Mechanics Division
NASA JSC Engineering Directorate

*JSC Engineering, Technology and Science (JETS) Contract*
*Engineering Department*
*Technical Brief*

**Date:** **December 12, 2017**
**Document Number:** **JETS-JE23-17-AFGNC-DOC-0066**
**Subject:** **Copernicus Version 4.5**

**Synopsis:**

Version 4.5 of the Copernicus spacecraft trajectory design and optimization system is released. This is an update to version 4.4 (which was released in October 2016). It includes various new features and bug fixes. The following memo provides a brief overview of the release. For more details, please consult the Copernicus User Guide.

**Prepared by:**

**Jacob Williams** **Date**
Senior Astrodynamics Engineer
GN&C and Aerosciences Section – JE23
JETS Engineering Department

**Approved by:**

**Robert G. Reitz** **Date**
Section Manager - JE23
GN&C and Aerosciences Section – JE23
JETS Engineering Department

**Approved by:**

**William H. Schoolmeyer** **Date**
Division Technical Manager (EG)
JETS Engineering Department

# Contents

# List of Figures



# List of Tables

# 1 Introduction

Copernicus [1] is a spacecraft trajectory design and optimization software application developed at the NASA Johnson Space Center (JSC). The latest Copernicus release (version 4.5) is an update to version 4.4 (which was released in October 2016). JSC makes Copernicus available free of charge to other NASA centers, government contractors, and universities, under the terms of a US government purpose license. Organizations interested in obtaining Copernicus should contact:


Technology Transfer and Commercialization Office
NASA Johnson Space Center
2101 NASA Parkway (Mail Code: AO5)
Houston, Texas 77058
Phone: (281) 483-3809
E-mail: jsc-techtran@mail.nasa.gov
URL: https://software.nasa.gov/software/MSC-25863-1


# 2 New Features

The following sections describe the significant new features included in the release.

## 2.1 New Data Output Options

Significant internal changes were made to how the segment data is exported to the files. The new GUI dialog is shown in Figure 1. Output files are now cached internally and generated all at once after the segment is propagated. This has the following effects:

- The output files are now generated much faster than before. For example, on one of the development machines, a 30-day propagation (with data file generation) of a low-earth orbit using an 8x8 gravity model, using DDEABM with a tolerance of $1 \times 10^{-12}$ took about 9 seconds with v4.4, and now takes about 4 seconds.
- The various output formats (CSV, JSON, SPK, and HDF5) can now be generated independently of each other. The user can select any combination (or none at all) to be generated. Formerly, for example, the JSON and SPK files were generated by reading the data from the CSV file. This results in increased speed when generating the non-CSV output files.
- Reversing the output files (for segments where $\Delta t < 0$) no longer requires reading and resaving each file. The data is now written in reverse order immediately.

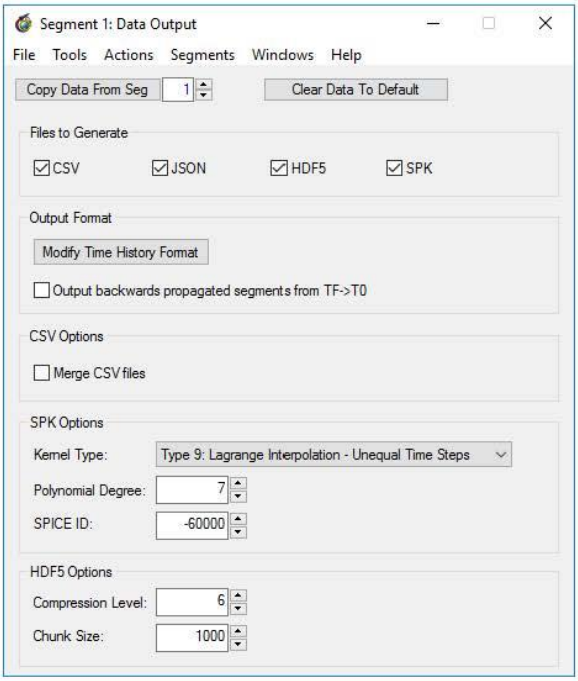



Figure 1: New Segment Data Output Dialog. The dialog now allows for independent generation of each of the output file formats (only the ones checked are generated). Also added was a new HDF5 file format.

A new option has been added to export the segment data in HDF5 [2] binary format. The compression level and chunk size can be specified in the GUI. The data in the HDF5 file is identical to what is stored in the CSV file for each segment. Each column in the CSV file corresponds to a group in the HDF5 file that contains the data and the column label datasets (see Figure 2). The files can be read with any of the available HDF5 interfaces (such as `h5py` in Python). See Figure 3 for an example of reading the Copernicus-produced HDF5 files with Python and converting them to CSV format.

Other data output changes include:

- Added a new command line flag (`-outfilesdir`) to generate the output files in a user-specified directory. Formerly, all output files were always generated in the same directory as the input deck. Redirecting them to a different location may be useful for scripting purposes.
- Increased the maximum number of time history variables in a segment from 100 to 256. In a future release, the upper limit will be eliminated entirely.
- If an error occurs during segment propagation, none of the output files are generated.
- In the JSON segment output files, a column of data is now printed on one row (without line breaks) to make the file a bit smaller and more human readable.

## 2.2 New Tools

Some new dialogs were added to the Tools menu:

- A "SPICE Tools" submenu was added. It contains two new dialogs:
  - A new eclipse finder dialog (shown in Figure 4). This is simply a wrapper for the SPICE `gfoclt()` routine [3].
  - A new time transformation dialog (shown in Figure 5). This is simply a wrapper for the SPICE `str2et()` and `timout()` routines [4, 5].
- A new "State Transformation Tool" was added (shown in Figure 6). This can be used to display the data ($\mathbf{R}$, $\dot{\mathbf{R}}$, $\mathbf{r}_{trans}$, and $\mathbf{v}_{trans}$) needed to transform a Cartesian state $[\mathbf{r}_i, \mathbf{v}_i]$ from one Copernicus frame to another using the equations:

$$\begin{bmatrix} \mathbf{r}_f \\ \mathbf{v}_f \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \dot{\mathbf{R}} & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{r}_i \\ \mathbf{v}_i \end{bmatrix} + \begin{bmatrix} \mathbf{r}_{trans} \\ \mathbf{v}_{trans} \end{bmatrix} \tag{1}$$

- A new dialog was added that shows all available units for each variable type.

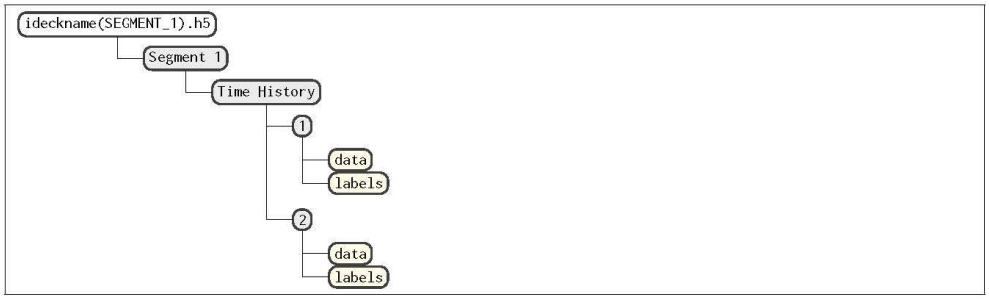



Figure 2: Copernicus HDF5 File Structure. Each "column" of data is a group (within the segment "Time History" group) that contains two datasets: "data" (which can be a vector of real numbers or strings), and "labels" (a single string containing the variable name). In the future, other groups may also be included within the segment group.

```python
import h5py
import pandas as pd
def hdf52csv(filenamein, filenameout):
    """ Convert a Copernicus HDF5 file to a CSV file """
    data = h5py.File(filenamein,'r')  # read the hdf5 file
    df = pd.DataFrame()    # create output pandas structure
    # get the data by column
    # note: currently there is only one segment
    # and time history group per file
    for seg in data:
        for th in data[seg]:
            num_cols = len(data[seg][th])
            for i in range(num_cols):
                c = str(i+1)  # column number
                label = data[seg][th][c]['labels'][0].decode('utf-8')
                if (isinstance(data[seg][th][c]['data'][0],bytes)):
                    df[label] = [d.decode('utf-8') \
                                 for d in data[seg][th][c]['data']]
                else:
                    df[label] = data[seg][th][c]['data']
    # write it as a csv file
    df.to_csv(filenameout,index_label=False,index=False)
```

Figure 3: Example Python Code for Conversion of HDF5 to CSV. This routine uses the `h5py` and `pandas` libraries and is specific to the file layout produced by Copernicus (see Figure 2).

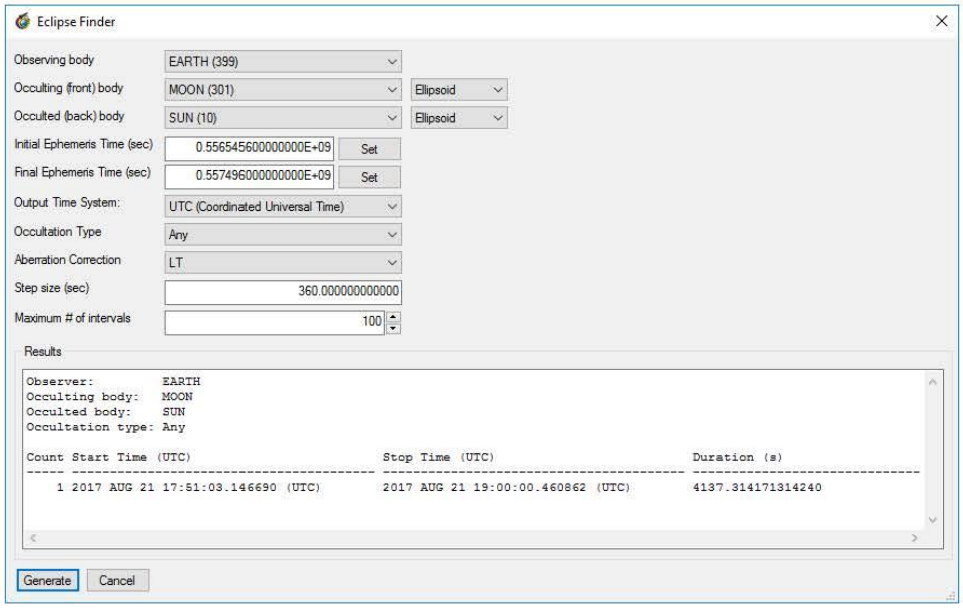



Figure 4: New Eclipse Finder Dialog. This can be used to find the epochs of occultation events among SPICE ephemeris objects.

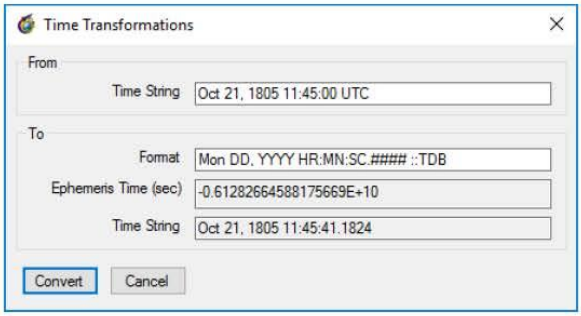



Figure 5: New Time Transformation Dialog. This can be used to quickly convert between different string representations of date and time.

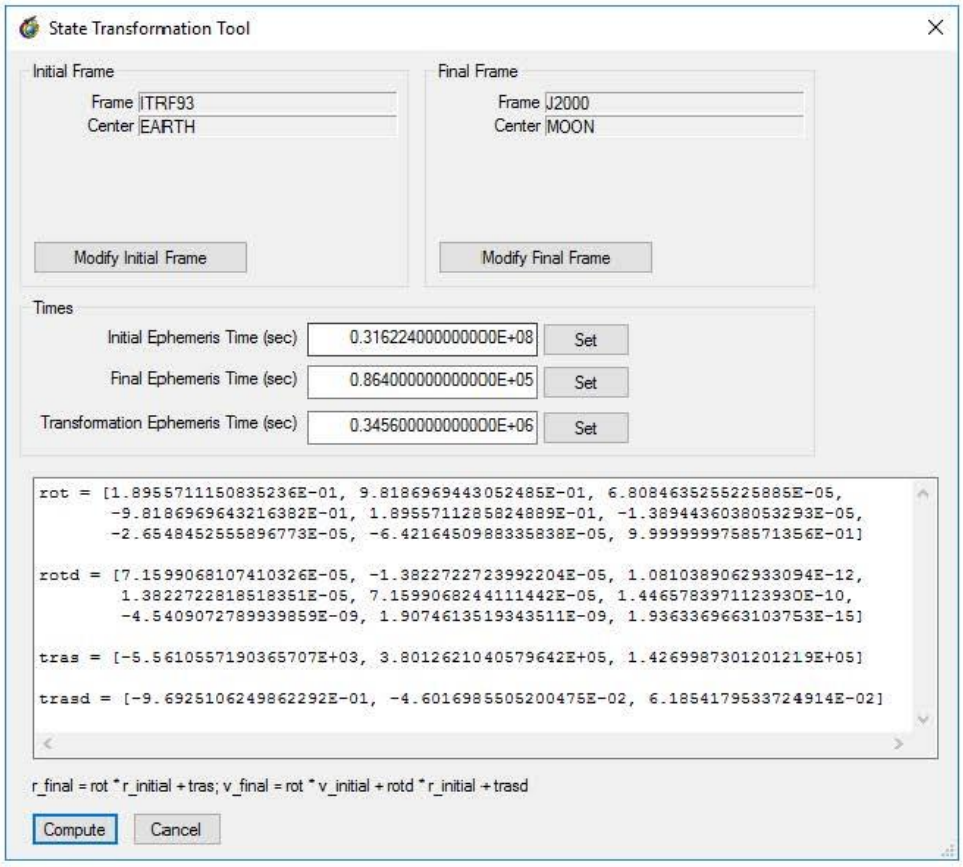



Figure 6: State Transformation Dialog. This dialog can be used to display the transformation matrices used by Copernicus to transform a Cartesian state from one frame to another. The user inputs the initial frame and epoch, the final frame and epoch, and the epoch at which to perform the transformation.

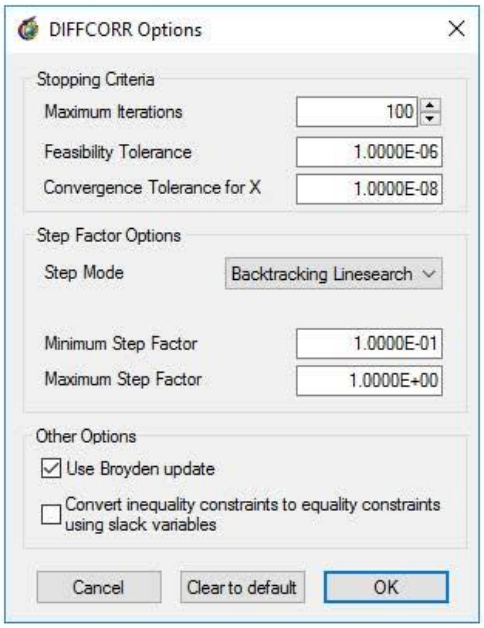



Figure 7: Updated DIFFCORR Dialog. The "Backtracking Linesearch" step mode and the Broyden update are new options in this release.

## 2.3 Updated Differential Corrector Options

The release includes some new options for the differential corrector solution method (DIFFCORR), which is a basic nonlinear equation solver. See Figure 7 for the updated dialog. The new additions are:

- A new backtracking line search option has been added for computing the step factor. When this mode is selected, a backtracking line search (minimizing the sum of the squares of the constraint violations) is used to determine the step size to take at each iteration.
- A Broyden update option [6] was also added, which uses a secant method to estimate the Jacobian matrix, rather than computing it using finite differences. The Jacobian computed in this manner is less accurate but much faster to compute, so the iterations will proceed faster but more may be required for convergence (this is the same type of algorithm used in the BROYDEN solution method).

## 2.4 Plugin Updates

Updates related to Copernicus plugins in this release include:

- DLL plugins can now access a small set of SPICE routines[1]. This allows for plugins to access data from the SPICE pool (such as frames and gravitational parameters). This requires linking the plugin with the provided `plugin-support` library. The full set of available routines are listed in Table 1. In the future, the set of available routines may be expanded. Note that it is not "safe" for plugins to add or remove pool variables, or change the loaded SPICE kernels, so those routines are not available.
- Also added a new option for plugins to inherit/push directly from/to the six initial state segment variables without doing a state transformation. This may be useful in some cases where the user knows what they are doing. The complete list of segment state plugin variables is shown in Table 2. The new variables (`STATEVAR1`, etc.) are scalar plugin variables. It is important to note that, when using these variables, unit conversions are performed like any other variable, but unlike the full `STATE` variable specification, no coordinate frame or parameterization transformation is done. For example, when pushing to `STATEVAR1`, the value is directly inserted into the first state variable as though the user had typed it directly in the GUI "State" dialog.
- When loading a config file associated with an existing ideck, Copernicus will now strip away just the plugin name from the file name and use that to construct the new config file name. This prevents unusually long file names when a user does this.

[1] `https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/spicelib/index.html`

Table 1: SPICE Routines Available in DLL Plugins

| Name | Description |
|---|---|
| bodc2n | Body ID code to name translation |
| bodn2c | Body name to ID code translation |
| bodvcd | Return d.p. values from the kernel pool |
| bodvrd | Return d.p. values from the kernel pool |
| cidfrm | Center ID to frame id and name |
| dtpool | Data for a kernel pool variable |
| et2lst | ET to local solar time |
| failed | Error status indicator |
| frmchg | Frame change |
| gcpool | Get character data from the kernel pool |
| gdpool | Get d.p. values from the kernel pool |
| getmsg | Get error message |
| gipool | Get integers from the kernel pool |
| gnpool | Get names of kernel pool variables |
| namfrm | Frame name to frame id |
| occult | Find occultation type at time |
| pgrrec | Planetographic to rectangular |
| phaseq | Phase angle quantity between bodies centers |
| plnsns | Planetographic longitude sense |
| prop2b | Propagate a two-body solution |
| recpgr | Rectangular to planetographic |
| refchg | Reference frame change |
| reset | Reset error status |
| spkez | S/P Kernel, easy reader |
| spkgeo | S/P kernel, geometric state |
| spkgps | S/P kernel, geometric position |

Table 2: Valid cop_var State Choices for Plugins

| Variable Type | cop_var String | Description | Valid Nodes |
|---|---|---|---|
| State | STATE | Full segment state variable structure | $t_0^-, t_0^+, t_f^-, t_f^+$ |
| | STATEVAR1 | First segment state variable | $t_0^-$ |
| | STATEVAR2 | Second segment state variable | |
| | STATEVAR3 | Third segment state variable | |
| | STATEVAR4 | Fourth segment state variable | |
| | STATEVAR5 | Fifth segment state variable | |
| | STATEVAR6 | Sixth segment state variable | |

- Failure to load a plugin config file is now a fatal error in command-line mode (formerly, Copernicus would print warnings, remove the plugin, and continue).
- The plugin dialogs are now separate dialogs accessible in the Mission Config window.
- Arrays of integers in plugin info structures are now displayed in the treeview (e.g., the group list). All info variables are now displayed (the modify button is disabled for the ones that should not be modified).
- A new `plugin-support` library is now included with Copernicus that can be linked with a user-created DLL plugin. The library includes the following modules that are useful for building DLL plugins:
  - `dll_message_module` – Allows the DLL to send messages to the input deck log file (which is also displayed in the GUI).
  - `spice_module` – Required for the DLL plugin to access the SPICE environment in Copernicus.
  - `json_module` – Required for all DLL plugins to communicate with Copernicus (see Chapter 17 in the Copernicus User Guide for details).
- The plugin input and output variable grids now have an additional column to display the variable types.

## 2.5 Frame Plugins

A new feature has been added to Copernicus to specify custom coordinate frames via DLL plugins. These plugins are different from the mission attribute plugins, and only exist to define the transformation between the `J2000` frame and the user-defined frame. The DLL must export the transformation subroutines with the exact interfaces shown in Figure 8. These subroutines return the $3 \times 3$ $\mathbf{R}$ (and optionally the $\dot{\mathbf{R}}$ and $\ddot{\mathbf{R}}$) rotation matrices (see Equation 1).

A frame plugin DLL is configured using a JSON config file, which is specified by adding the file name to the SPICE pool variable `COP_PLUGIN_FRAMES` in a PCK file. Any number of frames can be added in this way. For example:

```
COP_PLUGIN_FRAMES += ('example-frame-plugin/bin/frameplugin.json')
```

This path must be a subdirectory of the `<Copernicus>/support_files/plugins/frames` directory. Frame plugin config files cannot be in any other location. An example config file is shown here:

```
{
  "name": "MY_NEW_FRAME",
  "dll": "frameplugin.dll",
  "id": 1401805,
  "is_inertial": true
}
```

The frame plugin config file contains the following variables:

- `name` – The name of the frame (this will be the name listed in the various frame menus in the GUI). The name should not be the name of any frame already in Copernicus, or one of the built-in SPICE frames [7]. If multiple user-defined frames are to be loaded in the same input deck, care must be taken to ensure that they each have a unique name.
- `id` – A unique ID code that Copernicus will use to identify the frame. It must not be the ID code of any frame already in Copernicus, or one of the built-in SPICE frames. According to the SPICE documentation [7], the range 1400000 to 2000000 has been set aside as ranges of Frame IDs that can be used freely by SPICE users without fear of conflict with "officially recognized" frames. It is recommended to use a value in this range. If multiple user-defined frames are to be loaded in the same input deck, care must be taken to ensure that they each have a unique ID code.
- `dll` – The name of the frame DLL plugin. This must have the extension `.dll` on Windows and `.so` on Linux (Copernicus will select the proper extension for the platform). The file must be in the same directory as the corresponding config file. In the example above, the DLL file is located at `<Copernicus>/support_files/plugins/frames/example-frame-plugin/bin/frameplugin.dll`.
- `is_inertial` – An optional parameter that indicates if the frame is inertial (i.e., non-rotating). For inertial frames, the $\dot{\mathbf{R}}$ and $\ddot{\mathbf{R}}$ matrices are not used, so the transformation routines will be called without these arguments. If this variable is not present, its value is assumed to be `false` (i.e., a rotating frame is assumed).

Frame plugins must be compiled with the Intel Fortran compiler (to avoid issues, the same version used to compile Copernicus is recommended). An example project is included in the release.

## 2.6 New Ramp Control Law

A new ramp control law option has been added for finite burn engines. The new control law uses cubic spline equations to transition from the initial and final values to the ramp phase (see Figure 9). It can be selected in the finite burn engine options dialog (shown in Figure 10). Consider the

```
subroutine j2000_to_frame(et, rot, istat, rotd, rotdd)
!DEC$ ATTRIBUTES DEFAULT, DLLEXPORT, ALIAS:"j2000_to_frame" :: j2000_to_frame
real(wp),intent(in) :: et  !! ephemeris time (TDB sec)
real(wp),intent(out),dimension(3,3) :: rot !! R matrix
integer,intent(out) :: istat  !! status code (0=no errors)
real(wp),intent(out),dimension(3,3),optional :: rotd !! Ṙ matrix
real(wp),intent(out),dimension(3,3),optional :: rotdd !! R̈ matrix
end subroutine j2000_to_frame_interface
```

(a) Subroutine to transform from `J2000` to the user-defined frame.

```
subroutine frame_to_j2000(et, rot, istat, rotd, rotdd)
!DEC$ ATTRIBUTES DEFAULT, DLLEXPORT, ALIAS:"frame_to_j2000" :: frame_to_j2000
real(wp),intent(in) :: et  !! ephemeris time (TDB sec)
real(wp),intent(out),dimension(3,3) :: rot !! R matrix
integer,intent(out) :: istat !! status code (0=no errors)
real(wp),intent(out),dimension(3,3),optional :: rotd !! Ṙ matrix
real(wp),intent(out),dimension(3,3),optional :: rotdd !! R̈ matrix
end subroutine frame_to_j2000_interface
```

(b) Subroutine to transform from the user-defined frame to `J2000`.

```
subroutine get_status_message(istat,msg)
!DEC$ ATTRIBUTES DEFAULT, DLLEXPORT, ALIAS:"get_status_message" :: get_status_message
integer,intent(in) :: istat  !! a status code
character(len=:),allocatable,intent(out) :: msg !! the error message
end subroutine get_status_message_interface
```

(c) Subroutine to retreive an error message string for the integer `istat` error code returned by the other two routines.

Figure 8: Frame Plugin Subroutine Interfaces. A frame plugin must include the `j2000_to_frame()` and `frame_to_j2000()` subroutines. The `get_status_message()` routine is optional. The `!DEC$` compiler directives are necessary on Windows to indicate that the routines are exported by the DLL.

following example thrust ramp parameters:

$$\begin{aligned} T_0 &= 10 \text{ N} \\ T_f &= 100 \text{ N} \\ \%T_0 &= 40\% \\ \%T_f &= 40\% \\ \tau &= 360 \text{ sec} \\ \Delta t &= 1 \text{ hr} \end{aligned} \tag{2}$$

The resultant thrust history is shown in Figure 11. In the cubic spline law, the $\tau$ parameter is $1/2$ the time to transition from the initial value to the ramp phase, and $1/2$ the time to transition from the ramp phase to the final value. The spline segments are constructed to be tangent at both ends to the perfect tracking curve (if no time constant was used). The cubic spline law is guaranteed to begin and end on the specified values, whereas the original "first order system" law is not (depending on the value of $\tau$). Note that this is enforced by internally reducing $\tau$ if the user-specified value cannot achieve the transition in the time allowed.

## 2.7 New Command Line Options

Various new command line options are included in the release (they are shown in Table 3). In addition, the following changes were made:

- Added the SPICE and HDF5 version numbers to the `-v` command line help display.
- Copernicus will now quit if an error is encountered in command-line mode when using the `-importx` command line argument (rather than continuing as before).
- Various updates to the command line help messages.

An example command-line usage is shown below. In this example, an input deck is solved using a pointmass field only, then the resultant solution is exported back to the original input deck, and then it is solved again.

```
> copernicus -solve "example.ideck" -pointmass -exportx pointmass.json
> copernicus -solve "example.ideck" -importx pointmass.json -save
```

This scheme may be useful for some missions where a pointmass solution (which presumably can be generated faster) can be used as a good initial guess for the high-fidelity solution.

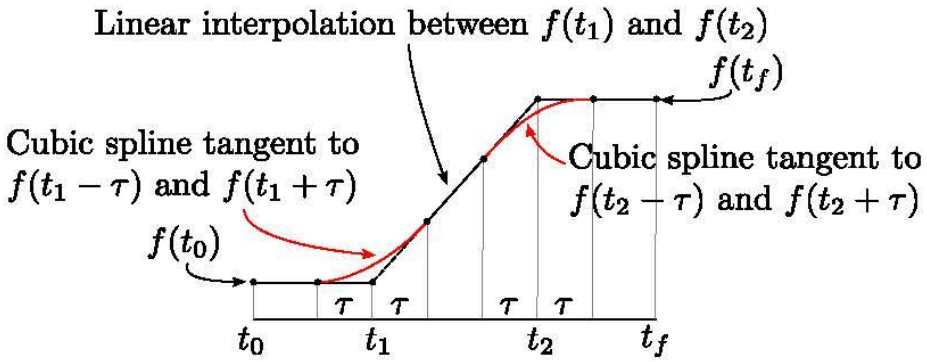



Figure 9: New Ramp Control Law. The control law determines the time history of an engine parameter (e.g., Thrust) from $t_0$ to $t_f$ (with a ramp phase from $t_1$ to $t_2$). In this figure, $f(t)$ (black) is the "perfect tracking" law, which can be smoothed using cubic polynomials at the corners using a time constant $\tau$ (red).

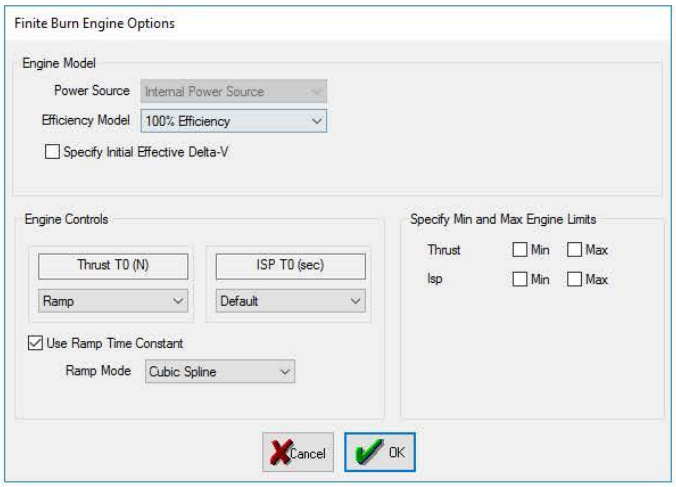



Figure 10: Finite Burn Engine Options Dialog. The new option is available in the "Ramp Mode" menu. The choices are "First Order System" (the original mode) and "Cubic Spline" (the new one).

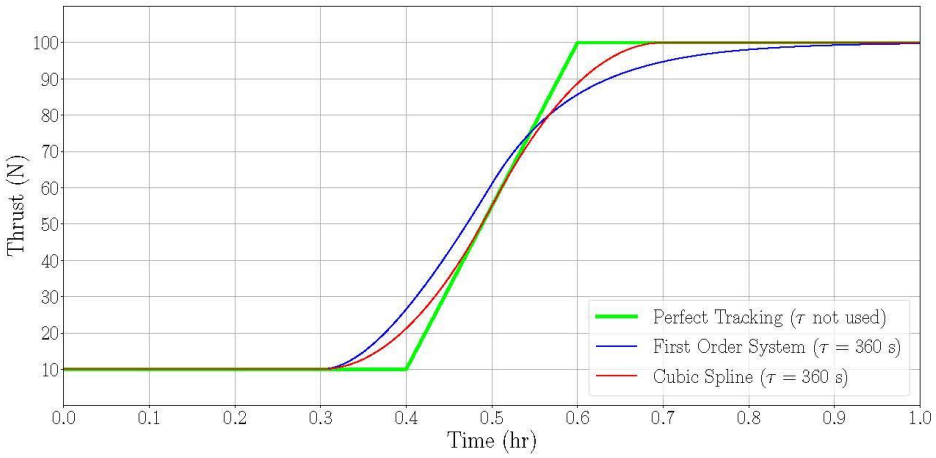
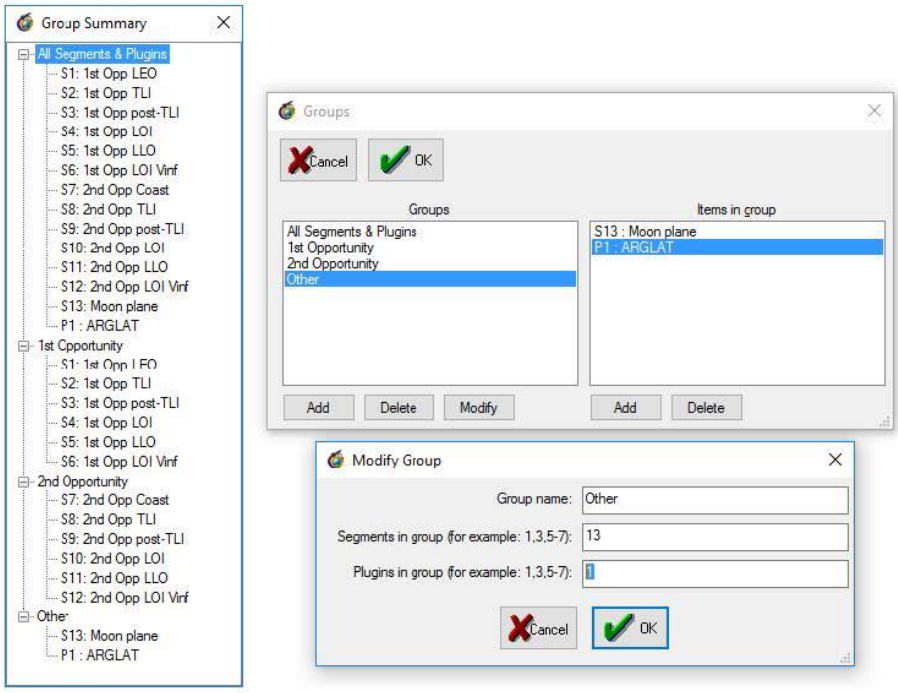



Figure 11: Two Engine Ramp Time Constant Modes. These results use the parameters in Equation 2. The original version in Copernicus is the "First Order System" option. The new mode is the "Cubic Spline" option. The new mode is always guaranteed to begin and end on the specified values, whereas the original is not (depending on the value of $\tau$).

Table 3: New Command Line Options

| Command Line Flag | Argument | Description |
|---|---|---|
| `-randomizexseed` | <integer value> | Specify the seed value for `-randomizex` |
| `-fps` | <integer value> | Specify the 3D graphics frame rate (the default is 30 frames per second, as before) |
| `-outfilesdir` | <directory name> | Send the segment output files to a specified directory |
| `-exportsparsity` | <file name> | Export the sparsity pattern to a JSON file (can be used only with the `-solve` flag) |
| `-update_epoch` | <epoch string> | Change the mission epoch while also adjusting all the segment $t_0$, $t_f$, and plugin time input variables so that the mission remains the same. The argument can be a string understood by the SPICE routine `STR2ET` [4] (e.g., ``1798 August 1, 18:20:00 UTC''), or a string indicating a segment time node from which to get the epoch (e.g., the initial time of segment 1 would be ``S1 T0'') |
| `-pointmass` | | Convert the central bodies of all segments to a pointmass gravity model |
| `-printcmd` | | Print the command used to invoke the program |




(a) Group Summary (b) Other Group Dialogs

Figure 12: Group Dialog Updates. Groups can now contain plugins in addition to segments. In this example, a plugin is included in the "Other" group. A group can contain any number of plugins, and must contain at least one segment.

## 2.8 New Group Options

The release includes the following changes to the groups feature:

- Added a "group summary" dialog to display all the groups and their contents (see Figure 12a).
- Groups can now also contain plugins (previous versions only allowed groups to contain segments, and plugins were global to all groups). Now, a group may contain any number of segments and any number of plugins, and only these included elements are included in the optimization problem. There must still be at least one segment in each group. See Figure 12b for an example. Note that the plugin list in the Mission Attributes dialog now only displays the plugins that are in the currently-selected group (this is the same behavior as the segment list).
- Some updates to the GUI display logic when modifying groups.

## 2.9 New Triaxial State Parameters

Three new state parameters were added to the geographic state parameterization:

- $h_T$ – triaxial ellipsoid altitude
- $\lambda_T$ – triaxial ellipsoid longitude
- $\phi_T$ – triaxial ellipsoid latitude

These parameters are analogous to the normal geodetic parameters ($h_D$, $\lambda$, and $\phi_D$), but are computed using an algorithm [8] that takes into account the triaxial shape of a body. In the SPICE pool, three distinct `RADII` values can be defined for celestial bodies. For example, for Europa:

```
BODY502_RADII = ( 1562.6 1560.3 1559.5 )
```

If we define the three radii values as:

$$a_x \equiv \texttt{BODY502\_RADII}(1) \tag{3}$$
$$a_y \equiv \texttt{BODY502\_RADII}(2) \tag{4}$$
$$b \equiv \texttt{BODY502\_RADII}(3) \tag{5}$$

then the Cartesian position vector components $[x,y,z]$ can be computed from the triaxial components using:

$$x = (\nu + h_T)\cos\phi_T \cos\lambda_T \tag{6}$$
$$y = \left(\nu(1-e_e^2) + h_T\right)\cos\phi_T \sin\lambda_T \tag{7}$$
$$z = \left(\nu(1-e_x^2) + h_T\right)\sin\phi_T \tag{8}$$

where:

$$e_e^2 = (a_x^2 - a_y^2)/a_x^2 \tag{9}$$

$$e_x^2 = (a_x^2 - b^2)/a_x^2 \tag{10}$$

are the first equatorial eccentricity and the first polar eccentricity, and:

$$\nu = \frac{a_x}{\sqrt{1 - e_x^2 \sin^2 \phi_T - e_e^2 \cos^2 \phi_T \sin^2 \lambda_T}} \tag{11}$$

is the radius of curvature in the prime vertical. Note that $a_x \geq a_y \geq b$ is assumed. An iterative method is used for the inverse transformation (Cartesian to triaxial parameters).

## 2.10 Miscellaneous Changes

The following other changes and new features are also included:

- Changed some real number formats in the GUI from `D` to `E`.
- If a SPICE body ID is encountered without a defined name (say, from a user-loaded SPK file), then Copernicus will now generate a name ("`BODY<ID>`") so it can be used without requiring the user to create a PCK file. It is still good practice under some circumstances to include a PCK file since that is the only way to specify other body properties such as the `RADII` or `GM` (a body with no associated PCK data will only have a defined name and no other properties).
- Added a new button in the SPICE dialog to export all the pool variables to a single kernel file.
- Added two new objective function modes, "RSS": $J = \sqrt{\sum_{i=1}^{n_j} (J_i^2)}$, and "Sum of Squares": $J = \sum_{i=1}^{n_j} (J_i^2)$, where $n_j$ is the number of objective function variables specified by the user, and $J$ is the objective function sent to the optimizer.
- Removed the old Subversion (SVN) version info from the input deck and command line help display (Copernicus is no longer using SVN for version control).
- Added a Python plotting option to the GUI for segment data output. This requires that Python be installed and visible in the system path. When clicking this button, a Python script is constructed to plot the data using Matplotlib, and then Python is called (note that the plot window blocks the rest of the GUI until it is closed). The new Python mode allows for panning and zooming which was not available before.
- Now displaying the full unit names in the state parameterization menus.

- Updated the linear solver routine to use LAPACK `DGESV()` and `DGELS()` rather than computing the inverse explicitly [9]. This routine is used by the Newton, Broyden, and DIFFCORR solution methods.
- Updated the toolbar icons.
- Segments and plugins are now subtabs under the main window "Mission Attributes" sidebar. Only the segments and plugins in the currently-selected group are shown in these fields.
- Added some more info to the PCK files that are created when creating a SPICE kernel in the GUI.
- Now saving real values with 17 digits of precision in the input deck and output files.
- Updates to ensure that the floating-point model compiler options are consistent for all platforms (Windows, Linux, and Mac). There were some inconsistencies in previous releases.
- Renamed "Auto" in the SNOPT gradient dialog to "Default" which is more descriptive.
- Added import and export optimization variable menus to the GUI. This feature now also includes the scale factors (in addition to the values).
- Separated all the text output tabs into individual dialogs.
- Added some additional error checking when exporting SPICE kernels.
- Added some new input deck examples.
- Updated to the latest SPICELIB release (N66).
- Now allowing any available Copernicus frame to be used as a spherical harmonic gravity frame (the user has to be aware of the frames appropriate to use for different bodies).
- Added the ability to change the default "printer friendly" 3D option via a `COP_DEFAULT_PRINTER_FRIENDLY` kernel variable. This allows the user to specify if the 3D starmap should be enabled by default when Copernicus starts up.
- Updated the help menu with the new GitLab link.
- Made the segment state dialog resizable to be consistent with the others.
- Increased the allowable string length for variable name and label fields.

# 3 Bug Fixes

The following bug fixes are included in this release:

- Fixed a bug where the unit conversions were not being done for the $v_y$ component of the geographic state parameterization (when the last three parameters were $[v_x, v_y, v_z]$).
- Fixed an issue where a stale Copernicus sparsity pattern was causing an "invalid input" error from SNOPT under some circumstances.
- Fixed an SNOPT bug where it would crash if the sparsity pattern had all 0's in a row.
- Fixed a GUI issue where changing the solution method did not update the simple bound columns in the plugin input variable grid.
- Copernicus now checks to make sure the user-specified `inherit_node` is valid in a plugin config file for inherits and pushes (formerly it would silently ignore some and crash on others).
- Fixed an issue where the absolute path to a SPICE kernel could be inserted into the input deck when it does the filename conversions on Linux. Note that this did not cause any problems, since Copernicus would normally fix it anyway.
- When changing the selected group in the Algorithms dialog, the selected segment and plugin is now updated if necessary.
- Fixed an issue where the freeze/unfreeze menu label was not always right for a newly-opened dialog.
- Fixed an issue where adding an OCT variable to the objective function would crash Copernicus.
- Fixed a glitch where the Mission summary file display was not updated in all cases.
- Fixed a bug where using the "scale norm of x vector" menu item could sometimes produce a negative scale factor, which would be displayed in the GUI as "*****".
- Fixed a bug where attempting to save an input deck that was read only would crash Copernicus.
- Fixed a typo in the DIFFCORR dialog (changed "Maximum" to "Minimum"). See Figure 7.
- Fixed an issue where, in some situations, only a maximum of 999 segments were being accounted for (for example, when plotting the trajectories).
- Fixed an issue where pushes to an initial segment state from a plugin may have produced an incorrect frame transformation if either frame was a rotating frame.

- Fixed an issue where clicking a checkbox in the $\Delta v$ to finite burn dialog would cause the other fields to reset.

# 4 Python Interface

The Copernicus/Python interface (CopPy) is included in the release in the "python" directory. Many updates have been made since the last release. They are described in the following sections.

## 4.1 Renumbering Inherits and Constraints

Previously, when using CopPy, if segments were inserted into or deleted from an input deck any remaining segment-linked inheritances and constraints were left as is. The CopPy module now increments/decrements the segment linkages appropriately to maintain linkages as they existed before the modification, unless those linkages point to the segment that was deleted. Those linkages remain as is. A second modification to the segment insertion software allows the user to copy multiple segments as a block from one input deck to another. Any segment linkages internal to this copied block are maintained and renumbered according to their location in the destination input deck. Any segment linkages that point to segments external to the copied block can be reassigned by the user. In this way, a block of segments can be hooked up to the destination input deck at the time the segments are inserted (see Figure 13 for an example).

## 4.2 Input Deck Differencing

Because the Copernicus input decks are structured in segments, using a standard text diff can sometimes obscure the actual differences. To address this, a capability was developed to evaluate the differences in two input decks on a segment by segment basis. This capability uses the segment name as the unique identifier and matches segments accordingly. The software evaluates whether segments have been added or deleted and evaluates the specific line differences between matched segments.

The input deck differencing capability was designed with a few applications in mind:

- Evaluating differences between input decks (and their plugins) in a way that is easier to interpret than a basic text diff. To aid in this, a capability is included to convert a diff file to an HTML file that gives the user a more visual sense of the differences.
- Enabling data-lean storage of a large scan by saving one base input deck and a list of diff files such that no data is lost. To this end, the diff file identifies deleted segments by name and line numbers, repeats added segments in their entirety, and lists individual line differences between matched segments.

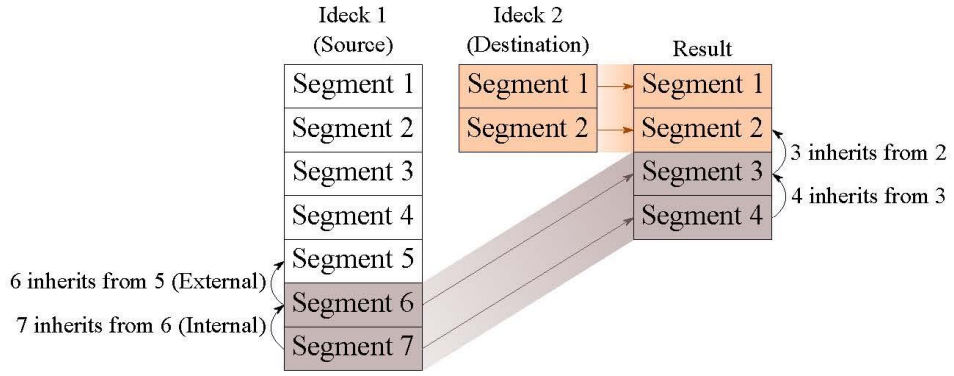



Figure 13: Appending Segments with CopPy. In this example, segments 6-7 from the source ideck are appended to the destination ideck. Segment 6 in the source ideck contains an external linkage to segment 5 (which the user maps to segment 2 in the destination ideck). The internal linkages are automatically renumbered.

- Enabling the reconstruction of an input deck by applying a diff file to a base input deck. This is the converse of the preceding application. If a diff file is generated to save storage space, a full input deck must be reconstructed in order for Copernicus to be able to open it.
- Streamlining the modification of input decks for a scan by enabling the application of a diff file to multiple input decks. This can be used for things like changing the optimizer, adding trajectory segments (like an abort), changing the reference epoch, etc.

## 4.3 Multiprocessing

The `solve()` method in the `copernicus_executable` class was modified to allow a user to take advantage of the multiple processor cores on their desktop machine or on a remote computing cluster. This multiprocessing capability allows the user to provide a multiprocessing job administrator of their choice (provided that it implements a `submit_job()` method).

## 4.4 Unit Tests

A suite of 1000 unit tests was added. These unit tests verify the expected operation of each of the methods in the CopPy module and use the Python unit test framework. The full suite of 1000 tests runs in less than 30 seconds and represents 95% code coverage by line as of this writing.

## 4.5 Miscellaneous Changes

The following other changes have been made since the 4.4 release of CopPy:

- Updated `check_solution()` to also check the optimization variable bounds. Also, this method can now return more information (by using the optional `return_all` argument).
- Added `get_group()` method.
- Added an optional `timeout` argument for `solve()` and `propagate()`. Also, `run()` now returns the exit code and any exceptions raised from `check_call`.
- Added experimental support for the future JSON input deck format. Note that CopPy can create and manipulate these files, but Copernicus can not yet open or save them. Also note that the final format of this file may change somewhat from its current form.
- Some updates in support of plugins:
  - When reading input decks, the plugin config structures (if present in external files) are now also read and added to the structure.
  - A `save()` is now required to update the plugin configs (and they are now always saved when `save()` is called).
  - The behavior of some of the plugin-related routines has changed. Also, it is no longer allowed to use the config file names to get/set plugin config data.
  - The plugin indices are now 1-based to be consistent with the segment numbering.
  - Added a new `append_plugin()` method.
- Added some missing get/set methods for various segment variables.
- Added new `set_seg_name()` and `set_seg_trajectory_color()` methods.
- Added support for `f90nml` versions $\geq$ v0.18.
- Added support for the new Copernicus command-line arguments (see Section 2.7).
- Added new group-manipulation methods: `number_of_groups()`, `get_group_names()` and `create_group()` routines. Updated `set_seg_group()` so that now a segment can be removed from all groups.
- Added a utility routine to convert a segment HDF5 file to a CSV file (see Figure 3).

### 4.6 Bug Fixes

The release includes the following bug fixes:

- Fixed bugs when using `anglewrap` argument.
- Fixed a bug in `get_reference_epoch()`.
- Fixed a bug in `extract_iteration()`.
- Fixed some typos in various variable names and methods.
- Fixed an issue where some very long variable names were not being written properly to the input deck.

## 5 Matlab Interface

The Copernicus/Matlab interface is now discontinued and will no longer be updated or distributed with Copernicus. Users that require a scripting capability for Copernicus are encouraged to use the Python interface (CopPy).

## 6 Developer Tools and Libraries

The following is a list of the various developer tools and libraries used to compile this version of Copernicus:

| Name | Version |
|---|---|
| Microsoft Visual Studio | 2013 Update 5 |
| .NET Framework | 4.5.2 |
| Intel Fortran Compiler | 2017 Update 4 |
| Intel Math Kernel Library | 2017 Update 3 |
| Winteracter | 10.10i |
| OpenSceneGraph | 3.5.4 (aae78b8) |
| OpenFrames | Revision e650d0b |
| SPICELIB | N0066 |
| SNOPT | 7.2-4 |
| JSON-Fortran | 6.1.0 |
| Bspline-Fortran | 5.3.0 |
| SLSQP | 1.0.0 |
| HDF5 | 1.8.18 |

# 7 Future Changes

In the next major release of Copernicus, it is planned to switch to a new JSON format for the input deck file. This change will enable new features going forward, and will also eliminate the separate plugin config files (they will be merged with the input deck). A backward-compatibility option will be provided so that existing `.ideck` files will be able to be opened and/or converted to the new format.

Another major change in the works is a complete rewrite of the Copernicus GUI using PyQt[2]. This will allow for a much more flexible and modern GUI and will be cross-platform for Windows, Linux, and Mac.

# 8 Platforms and System Requirements

Copernicus is a 64 bit application, and is available on three platforms (Windows, Linux, and Mac). To use the 3D graphics (on Windows and Mac), a graphics card that supports OpenGL and OpenSceneGraph [10] is required. Some users have reported that updates to their graphics card drivers were necessary to enable proper functioning of the 3D graphics (these should be downloaded from the graphics card manufacturer's website). The graphics card tested by the developers is an NVIDIA Quadro K5000, with driver version 341.21 (WHQL).

Platform-specific details are described in the following sections:

## 8.1 Windows

The Windows build of Copernicus is developed and tested on 64 bit Windows 7 and Windows 10. Microsoft's .NET Framework 4.5 (with the latest Service Pack) must be installed for Copernicus to work properly. The Windows build of Copernicus has also been known to work on a Mac using Apple Boot Camp, Parallels Desktop, or VMware to emulate Windows 7. However, this configuration is not tested by the developers and may also have issues.

## 8.2 Linux

The Linux command line build of Copernicus was built and tested on a 64 bit CentOS 6 system. It does not run on CentOS 5 since that system uses an earlier version of the GNU C Library (GLIBC) that is not compatible with the latest Intel Math Kernel shared library used by Copernicus (see Section 6 for the version numbers of the tools used to compile Copernicus).

The Linux build of Copernicus (the "cop" file in the `bin_v4-5_linux` folder) does not have a GUI, and can only be used in command-line mode. Otherwise, it has the same features as the Windows

[2] `https://www.riverbankcomputing.com/software/pyqt/intro`

build. In order to run on Linux, Copernicus requires various shared libraries from the Intel Fortran compiler to be installed [11]. These are located in the `bin_v4-5_linux` folder. To run Copernicus on Linux, this directory must be added to the system's shared library path. For example, by adding the following to your `.bash_profile` (or equivalent) login file:

```
export LD_LIBRARY_PATH="/path/to/Copernicus/bin_v4-5_linux:$LD_LIBRARY_PATH"
```

Note that Copernicus may experience problems running on a Linux system where the sysctl flag `kernel.exec-shield` is set to 2 or 3. If so, setting this flag to 0 or 1 may be necessary.

## 8.3 Mac

The 4.5 release includes the first experimental Mac build of Copernicus (which includes a full GUI). A screenshot is shown in Figure 14. The Mac build is an X11/OpenMotif[3] application that has the same capabilities as the Windows version (both GUI and command-line), with a few exceptions. To run the program, it is necessary that XQuartz 2.7.11[4] be installed on the system. It is also necessary to set environment variables to enable the application to locate the various runtime libraries present in the `bin_v4-5_mac` directory that Copernicus requires (e.g., the Intel, OpenFrames, and OpenSceneGraph `.dylib` files). For example, by adding the following to your `.bashrc` (or equivalent) login file:

```
export COP_BIN=/path/to/Copernicus/bin_v4-5_mac
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$COP_BIN
export DYLD_BIND_AT_LAUNCH
```

To launch the Copernicus GUI, simply `cd` to the `bin_v4-5_mac` directory from a Terminal and type `./cop`. The normal command-line flags also work, as on Windows and Linux. The Mac build of Copernicus is developed and tested on a Mid 2015 MacBook Pro running OS X 10.11.4. No other configurations have been tested.

It should be noted that support for X11/OpenMotif on the Mac is not great, and some GUI glitches will be encountered. The command-line version should work well, but the GUI will be much slower to refresh/redraw than the Windows version. There is currently nothing known that can be done about this (see Section 7 for the future direction of the Copernicus GUI). Any problems encountered when running in command-line mode should be reported to the developers. The following features are not yet available on the Mac:

- The "No Graphics" and "2D Graphics" tabs.
- HDF5 data export.
- Support for shared library plugins (it is possible that this works, but has not been tested).

[3] `http://www.winteracter.com/macmotif.htm`
[4] `https://www.xquartz.org/`

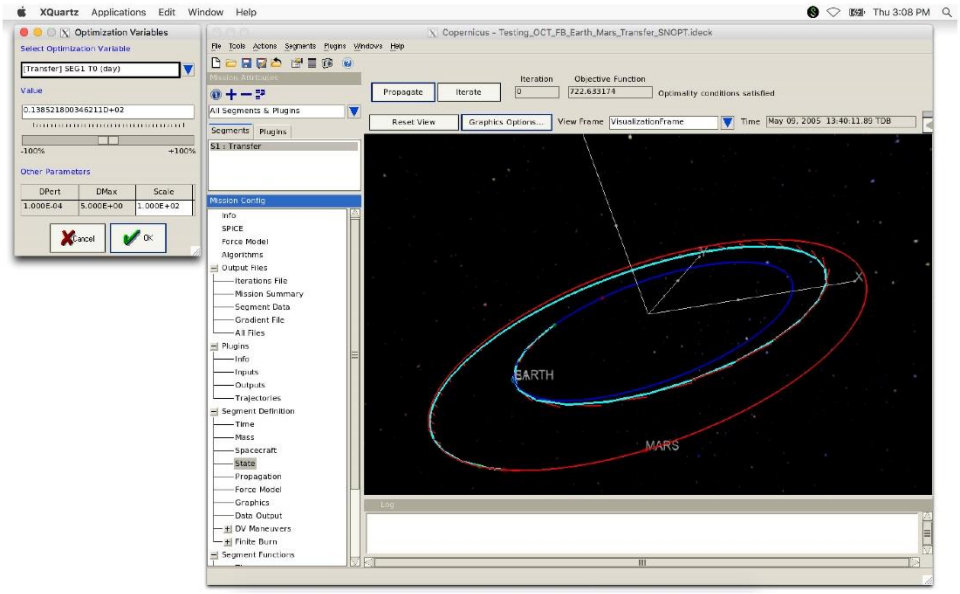



Figure 14: Copernicus on a Mac. The Mac build is an X11/OpenMotif application that runs in XQuartz. It contains almost all the same features as the Windows GUI.

# 9 Version Control and Bug Reporting

Version control, issue tracking and bug reporting for Copernicus is hosted at JSC on the following GitLab server:

`https://gitlab-fsl.jsc.nasa.gov/copernicus`

Note that an AGDL account is required to access this system, which is currently only available to NASA personnel. The Copernicus 4.5 release corresponds to revision 1166149 in the `copernicus-devel` git repository. Other repositories available on this site include various plugin examples and CopPy. The CopPy repository is public on the system and users with access are encouraged to clone the repository and submit merge requests with any changes or updates they want to contribute. The version of the Python interface released with Copernicus 4.5 corresponds to revision f67b00b in the `CopPy` repository.

For users without access to the JSC network, bug reports for Copernicus and CopPy may be sent to Jerry Condon at `gerald.l.condon@nasa.gov` (NASA JSC).

# 10 Acknowledgments


The author wishes to acknowledge Jerry Condon, Roland Martinez, Dave Dannemiller, and Dan Murri for their support of this work. Development for this update to Copernicus was funded by NASA JSC under contracts NNJ13HA01C and NNJ12HB20C, and included support from the NASA Engineering & Safety Center (NESC).


# Acronym List

**AGDL** Advanced Guidance, Navigation and Control Development Laboratory

**CSV** Comma Separated Values

**DLL** Dynamic-Link Library

**GLIBC** GNU C Library

**GUI** Graphical User Interface

**HDF5** Hierarchical Data Format

**HTML** Hypertext Markup Language

**JSC** Johnson Space Center

**JSON** JavaScript Object Notation

| | |
|---|---|
| **NASA** | National Aeronautics and Space Administration |
| **NESC** | NASA Engineering & Safety Center |
| **PCK** | Planetary Constants Kernel |
| **SPICE** | Spacecraft, Planet, Instruments, C-matrix and Events |
| **SPK** | Spacecraft and Planet Kernel |
| **SVN** | Subversion |

# References


[1] C. Ocampo, "An Architecture for a Generalized Trajectory Design and Optimization System," in *Proceedings of the Conference: Libration Point Orbits and Applications* (G. Gómez, M. W. Lo, and J. J. Masdemont, eds.), pp. 529–572, World Scientific Publishing Company, June 2003. Aiguablava, Spain.

[2] The HDF Group, "Hierarchical Data Format," 1997-2017. `http://www.hdfgroup.org/HDF5/`.

[3] NAIF, "GFOCLT." `https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/spicelib/gfoclt.html`.

[4] NAIF, "STR2ET." `https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/spicelib/str2et.html`.

[5] NAIF, "TIMOUT." `https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/spicelib/timout.html`.

[6] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Mathematics of Computation*, vol. 19, no. 92, pp. 577–593, 1965.

[7] NAIF, "Reference Frames." `https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/req/frames.html`.

[8] M. Ligas, "Cartesian to geodetic coordinates conversion on a triaxial ellipsoid," *Journal of Geodesy*, vol. 86, pp. 249–256, Apr 2012.

[9] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, third ed., 1999.

[10] "OpenSceneGraph." `http://www.openscenegraph.org/`.

[11] "Redistributable Libraries for Intel C++ and Fortran 2017 Compilers for Linux," May 2017. https://software.intel.com/en-us/articles/redistributables-for-intel-parallel-studio-xe-2017-composer-edition-for-linux.

# Appendix B. Copernicus Release 4.6.0

*JSC Engineering, Technology and Science (JETS) Contract*
*Engineering Department*
*Technical Brief*

**Date:** **April 30, 2018**
**Document Number:** **JETS-JE23-18-AFGNC-DOC-0009**
**Subject:** **Copernicus Version 4.6**

**Synopsis:**

This memo documents the 4.6 release of the Copernicus spacecraft trajectory design and optimization system. This is an update to the 4.5 release, with several bug fixes, minor modifications, and the addition of a few new features (such as expanded reference frame options and a new JSON kernel file format). For more details, please consult the Copernicus User Guide.

**Prepared by:**

**Jacob Williams** **Date**
Senior Astrodynamics Engineer
GN&C and Aerosciences Section – JE23
JETS Engineering Department

**Approved by:**

**Robert G. Reitz** **Date**
Section Manager - JE23
GN&C and Aerosciences Section – JE23
JETS Engineering Department

**Approved by:**

**William H. Schoolmeyer** **Date**
Division Technical Manager (EG)
JETS Engineering Department

# Contents

# List of Figures



# List of Tables

# 1 Introduction

Copernicus [1] is a spacecraft trajectory design and optimization software application developed at the NASA Johnson Space Center (JSC). The latest Copernicus release (version 4.6) is an update to version 4.5 (which was released in January 2018). JSC makes Copernicus available free of charge to other NASA centers, government contractors, and universities, under the terms of a US government purpose license. Organizations interested in obtaining Copernicus should contact:

Technology Transfer and Commercialization Office
NASA Johnson Space Center
2101 NASA Parkway (Mail Code: AO5)
Houston, Texas 77058
Phone: (281) 483-3809
E-mail: `jsc-techtran@mail.nasa.gov`
URL: `https://software.nasa.gov/software/MSC-25863-1`

# 2 Bug Fixes

The following bug fixes are included in this release:

- A bug in the geographic state parameterization that had been introduced in the 4.5 release has been fixed. This bug caused incorrect state transformations when the first three parameters were [`r`, `RA_P`, `DEC_P`] (magnitude, right ascension, and declination of the position vector).
- Fixed a long-standing issue in the 3D graphics on Windows which would sometimes cause the program to become unresponsive. This would sometimes occur for missions with many segments or with specific graphics cards when the system was under heavy load.
- Fixed a bug where Copernicus could crash if a dialog was opened immediately after the program launched, but before the 3D graphics had finished initializing.
- Fixed various bugs in the computation of angular inequality constraints among segments. Note that the sign of angular constraint violations is now the opposite of before, in order to be consistent with non-angular constraints. Also updated angular constraint computations for the Differential Evolution (DE) solution method to account for angle wrapping.
- Segment SPK data file exporting now works in the Mac and Linux builds.
- Fixed an issue that caused SPK exporting to fail for segments with long names.
- A minor issue was corrected where the last row of the iterations file after calling the solution method was not being properly finished (it only had, for example, `"SNOPTA","EXITS WITH","1"`, and not the other empty columns). This was not a problem in Copernicus, but may have caused problems if the user was parsing this file with other tools.

- Fixed an issue in the SLSQP solution method that could cause the program to crash under some circumstances if the number of equality constraints was greater than the number of optimization variables.
- Fixed an issue where the 4.5 release was not properly identifying the line number if an invalid line was encountered in an input deck.
- Fixed an issue in the SPICE Pool Variables popup dialog where string variables were being truncated to 64 characters (SPICE allows up to 80 characters for string variables).
- Fixed a bug where the VF13AD solution file was not being generated.

# 3 New Feature: JSON Kernel Files

```
{
  "COP_GGM03C_GM": 398600.4356,  // Change the μ for GGM03C gravity model
  "BODY399_TEXTUREMAP": "Earth_with_clouds.png",  // Earth texturemap
  "BODY399_BODYFIXEDGEOGRAPHIC": false, // Use input frame for Earth geographic
  "+COP_PLUGIN_FRAMES": ["sofa/IAU_76_80.json",
                         "sofa/IAU_06_00A.json"]  // Custom frame plugins
}
```

Figure 1: JSON Kernel Example. This example shows how to add variables to the SPICE pool using a JSON file.

A new feature has been added to allow the option of specifying SPICE pool variables in JSON files. This is an alternative to the native text PCK file format supported by SPICE. The file format is simply a list of the pool variables and their values. Scalars and 1D arrays of reals, integers, character strings, and booleans are supported. An example is shown in Figure 1.

JSON kernels are loaded like normal kernels in the SPICE dialog, and Copernicus allows both types to be used in the same mission. The format is a standard JSON file [2], with the addition that comments are also supported (using the `//` syntax). The new format also supports adding new elements to an existing pool variable (analogous to the "+=" syntax in PCK files [3]). This is done by adding a "+" character to the beginning of the variable name (as shown with the `COP_PLUGIN_FRAMES` variable in Figure 1).

One advantage of Copernicus JSON kernels is that they are cross-platform (normal SPICE PCK files are not, thus requiring separate Windows and Mac/Linux version). Another advantage is that they can be easily created, parsed, and manipulated by other tools. In a future release of Copernicus, the default `Copernicus_Variables` kernel may be converted to the JSON format.

# 4 New Features: Reference Frames

Various updates were made in this release related to reference frames:

- Frame plugins can now use the `plugin-support` library to call SPICE routines and send messages to Copernicus (in the same why that normal DLL plugins can do this).
- Added two new optional routines that can be present in DLL frame plugins:
  - `initialize()` – called when the plugin is loaded.
  - `destroy()` – called when the plugin is unloaded.

  The interfaces to the frame plugin routines were changed to accommodate these new features (note that this does break backward compatibility for frame plugins created for use with the 4.5 release). The interfaces to all the frame plugin routines are shown in Figure 2. Note that all routines now have an integer `ID` code input, which corresponds to the unique ID code for the frame. This means that it is now possible for a single frame plugin DLL to manage multiple frames (it is up to the plugin to properly handle this, for example, by returning the proper transformation matrices based on the input ID code).
- Frame plugins can now be assigned to the `BODY_FIXED` frame associated with a celestial body.
- Added a new option to specify whether the geographic state parameterization uses the body's `BODY_FIXED` frame for geodetic elements or the input frame. By default, the `BODY_FIXED` frame is used for all bodies (this is the original behavior in Copernicus). To disable the default behavior and instead use the input frame for a certain body, you can now use a kernel variable such as (for the Earth):

  ```
  BODY399_BODYFIXEDGEOGRAPHIC = (0)
  ```

  If using this feature, it is up to the user to make sure that a frame is appropriate for use as a body-fixed frame for the specified body (Copernicus will not complain if an inappropriate frame is specified).
- Environment variables are now allowed in the specification of the location of frame plugin config files.
- Minor change to allow for more frames to be used in different circumstances. Any inertial SPICE frame can now be used for the mission's force base frame. Any non-inertial, non-2body frame can now be used as a gravity model or atmosphere frame. It is up to the user to choose the appropriate frames to use depending on the application.
- Added the `IAU_BENNU` frame to the set of IAU body-fixed frames recognized by Copernicus. This frame was added in the SPICELIB N66 release [4].

```
subroutine initialize(id,config,config_path,plugin_path,istat)
!DEC$ ATTRIBUTES DEFAULT, DLLEXPORT, ALIAS:"initialize" :: initialize
integer,intent(in) :: id  !! frame ID code
type(json_file),intent(inout) :: config  !! JSON data for config file
character(len=*),intent(in) :: config_path !! plugin config file path
character(len=*),intent(in) :: plugin_path !! plugin DLL file path
integer,intent(out) :: istat  !! status code (0=no errors)
end subroutine initialize
```

(a) Subroutine for plugin frame initialization (optional).

```
subroutine j2000_to_frame(id, et, rot, istat, rotd, rotdd)
!DEC$ ATTRIBUTES DEFAULT, DLLEXPORT, ALIAS:"j2000_to_frame" :: j2000_to_frame
integer,intent(in) :: id  !! frame ID code
real(wp),intent(in) :: et  !! ephemeris time (TDB sec)
real(wp),intent(out),dimension(3,3) :: rot !! R matrix
integer,intent(out) :: istat  !! status code (0=no errors)
real(wp),intent(out),dimension(3,3),optional :: rotd !! Ṙ matrix
real(wp),intent(out),dimension(3,3),optional :: rotdd !! R̈ matrix
end subroutine j2000_to_frame_interface
```

(b) Subroutine to transform from `J2000` to the user-defined frame.

```
subroutine frame_to_j2000(id, et, rot, istat, rotd, rotdd)
!DEC$ ATTRIBUTES DEFAULT, DLLEXPORT, ALIAS:"frame_to_j2000" :: frame_to_j2000
integer,intent(in) :: id  !! frame ID code
real(wp),intent(in) :: et  !! ephemeris time (TDB sec)
real(wp),intent(out),dimension(3,3) :: rot !! R matrix
integer,intent(out) :: istat !! status code (0=no errors)
real(wp),intent(out),dimension(3,3),optional :: rotd !! Ṙ matrix
real(wp),intent(out),dimension(3,3),optional :: rotdd !! R̈ matrix
end subroutine frame_to_j2000_interface
```

(c) Subroutine to transform from the user-defined frame to `J2000`.

```
subroutine get_status_message(id,istat,msg)
!DEC$ ATTRIBUTES DEFAULT, DLLEXPORT, ALIAS:"get_status_message" :: get_status_message
integer,intent(in) :: id  !! frame ID code
integer,intent(in) :: istat  !! a status code
character(len=:),allocatable,intent(out) :: msg !! the error message
end subroutine get_status_message_interface
```

(d) Subroutine to retrieve an error message string for the `istat` error code returned by the other routines.

```
subroutine destroy(id,istat)
!DEC$ ATTRIBUTES DEFAULT, DLLEXPORT, ALIAS:"destroy" :: destroy
integer,intent(in) :: id  !! frame ID code
integer,intent(out) :: istat  !! status code (0=no errors)
end subroutine destroy
```

(e) Subroutine for frame plugin destruction (optional).

Figure 2: Frame Plugin Subroutine Interfaces. A frame plugin must include the `j2000_to_frame()` and `frame_to_j2000()` subroutines. The others are optional. The `!DEC$` compiler directives are necessary on Windows to indicate that the routines are exported by the DLL.

# 5 Miscellaneous Changes

- Added the ability to specify the gravitational parameter ($\mu$) for spherical harmonic gravity models in a SPICE pool variable. This will overwrite the one in the gravity model CSV file. For example, changing the $\mu$ for the GGM03C model could be done in a text PCK kernel by adding the line:

  ```
  COP_GGM03C_GM = (398600.4356)
  ```

  Note that, for the Earth, the $\mu$ in the gravity model CSV file is consistent with the TDT time scale. Since Copernicus uses the TDB time scale for integration, the TDB $\mu$ above should be used for total consistency [5].

- Added the following additional SPICE routines to the plugin interface (i.e., the set of SPICE routines that can be called from within a DLL plugin): `et2lst()`, `str2et()`, `timout()`, `unitim()`, `deltet()`, `utc2et()`, `et2utc()`, and `ttrans()`. See the SPICE documentation for details on these routines [6].

- A change was made to the escaping logic for "/" characters in JSON files. The previous release would always escape these (producing "\/"), which was not really desirable in unit strings or parser equations. Since it is optional for this character to be escaped in JSON [2], Copernicus will no longer escape it.

- The segment name can now be obtained from an HDF5 segment output file. This is stored in the "name" dataset in a new "Attributes" group. In the future more information may be added to the "Attributes" group.

- When generating the segment time history output files, all existing output files associated with the input deck (`.CSV`, `.json`, and `.h5`) are first deleted before the new files are generated. Formerly, only the `.CSV` files were deleted.

- Updated the logic for locating a plugin DLL or script file. The input path specified in the plugin config file (which can contain environment variables using the "`${DIR}`" syntax) is now interpreted in the following three ways (in order of precedence):

  - Use the input path as an absolute path or path relative to the Copernicus executable.
  - Strip away the input path directories, and append only the file name (DLL or script) to the input deck directory. This allows for putting the plugin in the same directory as the input deck (similar to the fallback behavior for locating SPICE kernels). This was always the fallback behavior for DLL plugins, and is now the same for script plugins.
  - Append the input path in its entirety to the input deck directory. This option is new to this release, and allows for the specification of a plugin in a subdirectory relative to the input deck. For example, if the input deck is being stored in a git repository, the plugin could be imported from a git submodule, without having to specify the absolute path or use an environment variable to locate it.

- Added a new command line argument (`-onlypropselectedgroup`) which disables propagation for segments and plugins not in the currently-selected group.
- The `-nopropfrozen` command line argument now also applies to plugins (formerly, it only applied to segments).

# 6 Addendum to the 4.5 Release Memo

The build date in the "About Copernicus" window for the 4.5 release was not correct (the year should have been 2018, not 2017). The actual Copernicus 4.5 release was revision cab9282 in the JSC `copernicus-devel` git repository. The following additional changes were also included in the 4.5 release but were not mentioned in the release memo [7]:

- Fixed a bug where inherit/push operations for `SOC` plugin variables did not work properly.
- The PCK kernels `Copernicus_Variables.tpc` and `Copernicus_Variables.tpc.pc` were modified so that the starmap database is used by default rather than the starmap image file. See the header in that file for details. This applies when the "Use Printer-Friendly Colors" option is not checked in the 3D graphics options dialog.

# 7 Developer Tools and Libraries

Table 1 shows a list of the various developer tools and libraries used to compile this release of Copernicus.

# 8 Acknowledgments


The author wishes to acknowledge Jerry Condon, Dan Murri, Ravi Mathur, and Randy Eckman for their support and contributions to this work. Development for this update to Copernicus was funded by NASA JSC under contract NNJ13HA01C, and also included support from the NASA Engineering & Safety Center (NESC).


# Acronym List

**CSV** Comma Separated Values

**DE** Differential Evolution

Table 1: Developer Tools & Libraries

| Name | Version |
|---|---|
| Microsoft Visual Studio | 2013 Update 5 |
| .NET Framework | 4.5.2 |
| Intel Fortran Compiler | 2017 Update 4 |
| Intel Math Kernel Library | 2017 Update 3 |
| Winteracter | 10.10i |
| XQuartz | 2.7.11 |
| Open Motif | 2.3.6 patched |
| OpenSceneGraph | 3.5.4 (aae78b8) |
| OpenFrames | Revision e650d0b |
| SPICELIB | N0066 |
| SNOPT | 7.2-4 |
| JSON-Fortran | 6.3.0 |
| Bspline-Fortran | 5.3.0 |
| SLSQP | 1.0.2 |
| HDF5 | 1.8.18 |

**DLL** Dynamic-Link Library

**HDF5** Hierarchical Data Format

**IAU** International Astronomical Union

**JSC** Johnson Space Center

**JSON** JavaScript Object Notation

**NASA** National Aeronautics and Space Administration

**NESC** NASA Engineering & Safety Center

**PCK** Planetary Constants Kernel

**SPICE** Spacecraft, Planet, Instruments, C-matrix and Events

**SPK** Spacecraft and Planet Kernel

**TDB** Barycentric Dynamical Time

**TDT** Terrestrial Dynamical Time

# References


[1] C. Ocampo, "An Architecture for a Generalized Trajectory Design and Optimization System," in *Proceedings of the Conference: Libration Point Orbits and Applications* (G. Gómez, M. W. Lo, and J. J. Masdemont, eds.), pp. 529–572, World Scientific Publishing Company, June 2003. Aiguablava, Spain.

[2] "JSON Website." `http://www.json.org/`.

[3] NAIF, "SPICE Kernel Pool Required Reading." `https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/req/kernel.html`.

[4] NAIF, "What's New in SPICE." `https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/info/whatsnew.html`.

[5] D. D. McCarthy and G. Petit, eds., *International Earth Rotation and Reference Systems Service (IERS) Conventions (2003), IERS Technical Note No. 32*. Frankfurt am Main: Verlag des Bundesamts für Kartographie und Geodäsie, 2004. `https://www.iers.org/IERS/EN/Publications/TechnicalNotes/tn32.html`.

[6] NAIF, "Index of SPICELIB Functions." `https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/spicelib/index.html`.

[7] J. Williams, *Copernicus Version 4.5*. JSC Engineering, Technology and Science (JETS) Contract, NASA Johnson Space Center, Dec. 2017. JETS-JE23-17-AFGNC-DOC-0066.

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| 01/17/2019 | Technical Memorandum | |

**4. TITLE AND SUBTITLE**
Improvements to the Copernicus Trajectory Design and Optimization System for Complex Space Trajectories

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**
Murri, Daniel G.; Condon, Gerald L.; Williams, Jacob; Kamath, Anubhav H.; Eckman, Randy A.; Mathur, Ravishankar

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**
869021.05.05.02.15

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
NASA Langley Research Center
Hampton, VA 23681-2199

**8. PERFORMING ORGANIZATION REPORT NUMBER**
L-20994 NESC-RP-15-01097

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
National Aeronautics and Space Administration
Washington, DC 20546-0001

**10. SPONSOR/MONITOR'S ACRONYM(S)**
NASA

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**
NASA/TM-2019-220247

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Unclassified - Unlimited
Subject Category 16 Space Transportation and Safety
Availability: NASA STI Program (757) 864-9658

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
The purpose of this assessment was to develop updates and new features for the NASA Copernicus Spacecraft Trajectory Design and Optimization analysis tool (version 5.0) for application to NASA programs and projects. This report details significant upgrades that were made to Copernicus in support of the NESC assessment. These upgrades represent major new capabilities that have been added to the tool for support of a variety of NASA projects and missions.

**15. SUBJECT TERMS**
Copernicus, Trajectories; Space Launch System; Graphical User Interface; Python Interface to Qt

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | STI Help Desk (email: help@sti.nasa.gov) |
| U | U | U | UU | 85 | 19b. TELEPHONE NUMBER *(Include area code)*<br>(443) 757-5802 |