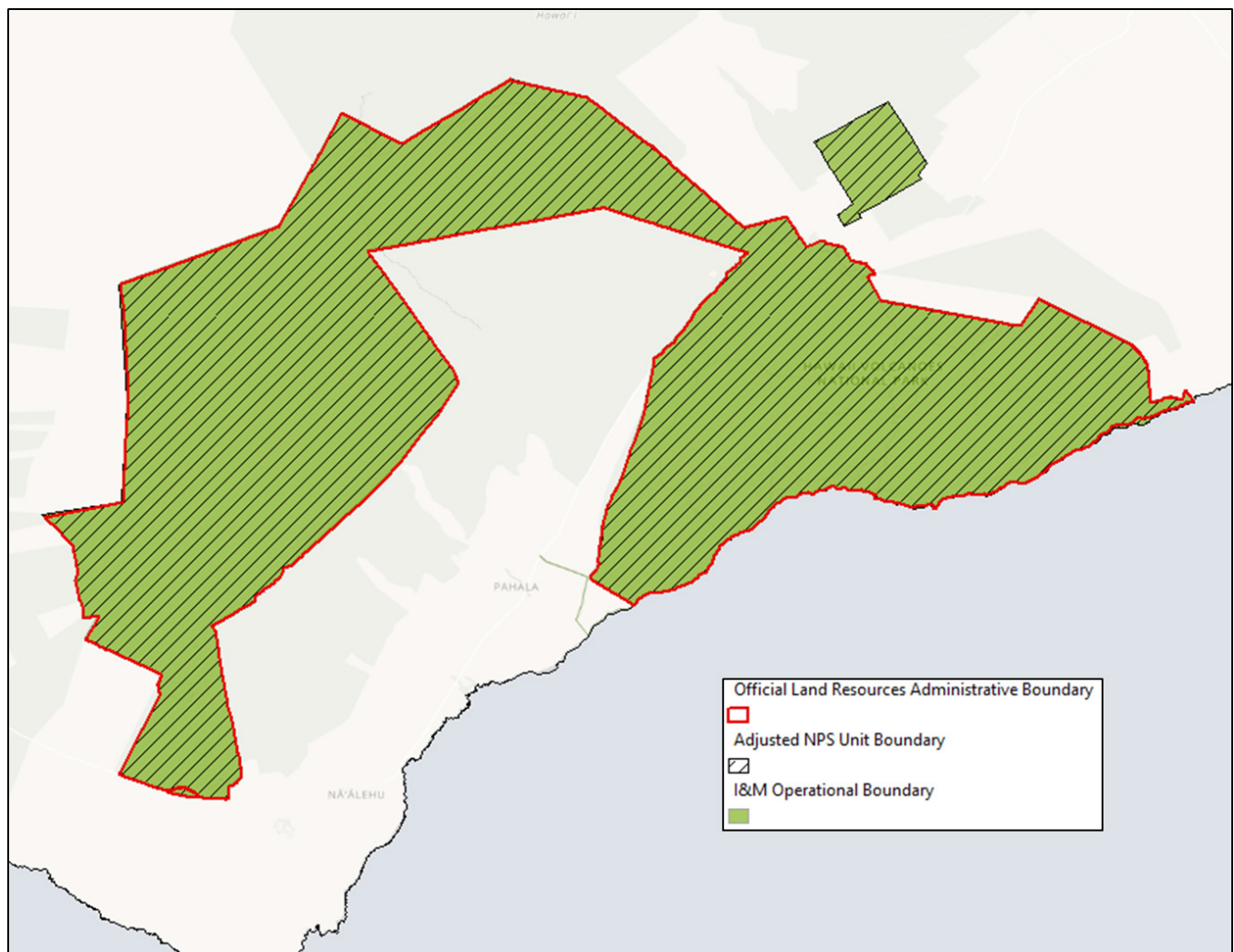




Unit Boundary Processing for National Park Service Units for Use in the National Park Environmental Settings Monitoring Protocol.

Fall 2018

Natural Resource Data Series NPS/NRSS/IMD/NRDS—2020/1253



ON THE COVER

Various boundary iterations for Hawaii Volcanoes National Park
Photography by NPS Inventory and Monitoring Division

Unit Boundary Processing for National Park Service Units for Use in the National Park Environmental Settings Monitoring Protocol.

Fall 2018

Natural Resource Data Series NPS/NRSS/IMD/NRDS—2020/1253

Michelle Kinseth,¹ Lisa Nelson,²

¹ Managed Business Solutions (MBS), a Sealaska Company
Contractor to National Park Service
Natural Resource Stewardship and Science
Inventory & Monitoring Division
1201 Oakridge Dr., Suite 150
Fort Collins, CO 80525

² National Park Service
Natural Resource Stewardship and Science
Inventory and Monitoring Division
1201 Oakridge Dr., Suite 150
Fort Collins, CO 80525

January 2020

U.S. Department of the Interior
National Park Service
Natural Resource Stewardship and Science
Fort Collins, Colorado

The National Park Service, Natural Resource Stewardship and Science office in Fort Collins, Colorado, publishes a range of reports that address natural resource topics. These reports are of interest and applicability to a broad audience in the National Park Service and others in natural resource management, including scientists, conservation and environmental constituencies, and the public.

The Natural Resource Data Series is intended for the timely release of basic data sets and data summaries. Care has been taken to assure accuracy of raw data values, but a thorough analysis and interpretation of the data has not been completed. Consequently, the initial analyses of data in this report are provisional and subject to change.

All manuscripts in the series receive the appropriate level of peer review to ensure that the information is scientifically credible, technically accurate, appropriately written for the intended audience, and designed and published in a professional manner.

The creation of data sets documented in this report followed peer reviewed methods described or referenced herein. Resultant data sets were reviewed to ensure consistency with documented data quality standards, as well as for usability and reproducibility.

Views, statements, findings, conclusions, recommendations, and data in this report do not necessarily reflect views and policies of the National Park Service, U.S. Department of the Interior. Mention of trade names or commercial products does not constitute endorsement or recommendation for use by the U.S. Government.

This report is available in digital format from the [Natural Resource Publications Management website](#). If you have difficulty accessing information in this publication, particularly if using assistive technology, please email irma@nps.gov.

Please cite this publication as:

Kinseth, M., and L. Nelson. 2020. Unit boundary processing for National Park Service units for use in the national park environmental settings monitoring protocol: Fall 2018. Natural Resource Data Series NPS/NRSS/IMD/NRDS—2020/1253. National Park Service, Fort Collins, Colorado.

Contents

	Page
Figures.....	iv
Tables.....	iv
Appendices.....	v
Abstract.....	vi
Background and Summary.....	1
Methods.....	2
Source Data	3
NPS Boundary Data from Land Resources Division	3
Alternate Boundaries.....	3
Updating Alternate Boundaries	4
Incorporate Boundary Changes from Parks and Networks	4
Incorporate Boundaries Absent from LRD Boundary Data	4
Incorporate Existing Alternate Boundaries	5
Creation of Adjusted NPS Unit Boundaries.....	6
Inactivation of Obsolete Adjusted NPS Unit Boundaries	7
Creation of I&M Operational Boundaries.....	8
Inactivation of Obsolete I&M Operational Boundaries	9
Additional Processing.....	9
Code Availability.....	9
Data Records.....	10
Data Quality Evaluation.....	12
Usage Notes	12
References.....	13

Figures

	Page
Figure 1. Generalized process flow for boundary processing.....	2

Tables

	Page
Table 1. Summary of all updated boundaries in output datasets Adjusted NPS Unit Boundaries, and I&M Operational Boundaries.....	3
Table 2. Status of boundaries within the affiliated areas and local boundary adjustments source data sets	4
Table 3. I&M park unit boundaries absent from the official Land Resources Division (LRD) administrative boundary dataset as of the September 30, 2018 LRD release	5
Table 4. Evaluation criteria summary of existing alternate boundaries.....	6
Table 5. Feature alteration code descriptions for output datasets.	6
Table 6. Feature alteration counts for adjusted NPS unit boundary datasets compared to input Land Resources Division source.....	7
Table 7. Counts of change reason from preceding version for adjusted NPS unit boundaries dataset.	7
Table 8. Active and inactive feature counts for adjusted NPS unit boundaries dataset.....	7
Table 9. Feature alteration counts for I&M operational boundaries datasets compared to input adjusted NPS unit boundary source dataset.	8
Table 10. Counts of change reason from preceding version for I&M operational boundaries dataset.	9
Table 11. Active and inactive feature counts for I&M operational boundaries dataset.....	9
Table 12. Overview of datasets and their provenance..	11

Appendices

	Page
Appendix A. Python Toolbox Code Affiliated Areas and Local Boundary Adjustments Verification	15
Appendix B. Python Toolbox Code Processing NPS Unit Bounds	21
Appendix C. Python Toolbox Code Processing I&M Operational Bounds.....	33
Appendix D. Additional Processing Script for Adjusted NPS Unit Boundaries	54
Appendix E. Additional Processing Script for IMD Operational Boundaries	59
Appendix F. Session and Version Information.....	65

Abstract

This data report documents the acquisition, preparation and creation of adjusted National Park Service (NPS) unit boundaries for use within the Inventory and Monitoring Program (I&M). The datasets are based on the NPS administrative boundary data release from the NPS Land Resources Division (LRD) as of September 2018 and supersede all previous releases of adjusted boundaries for use within the I&M Program.

The LRD boundaries are included, or in some cases are replaced, with I&M operational boundaries to create an integrated dataset of NPS boundaries that are used for a spectrum of I&M activities.

This integrated dataset is the foundation for several activities undertaken by the Inventory & Monitoring Program, including the national Environmental Settings protocol (DeVivo et al. 2018) and the IMD Inventory 2.0 program (DeVivo 2019), as well as several efforts being conducted by networks and parks. Boundary datasets documented in this report are intended only for use within the Inventory and Monitoring Program and may not include all parks, units, or other areas of interest to the NPS.

Background and Summary

The Inventory and Monitoring Program (I&M) monitors natural resources in many, but not all, park units across the National Park Service (NPS). Landscape dynamics, weather, and stream hydrology measures are monitored in areas inside and near parks with significant natural resources using methods outlined in the NPS Environmental Settings protocol (DeVivo et.al 2018). The areas monitored are defined by I&M operational boundaries. These operational boundaries may differ from the official boundaries of lands owned, managed, or authorized as a part of the NPS System and are not intended to be a replacement or correction of official administrative boundaries.

When the NPS Land Resources Division (LRD) publishes official administrative boundaries—which include only lands owned & operated by the NPS—the boundaries for parks of interest to I&M are reviewed and, if necessary, adjusted boundaries are created or updated. The results of these updates or additions is a data set containing I&M operational boundaries. In some cases, the boundaries used as the basis of monitoring of these parks differ from the official administrative boundaries as defined by LRD for a variety of reasons, including corrections to known errors in the LRD data, adjustments to boundaries that have not yet been incorporated into LRD data, or inclusion of (non-NPS) areas of interest within or near parks. In these instances, for the purposes of inventory and monitoring efforts carried out by I&M, a boundary provided by the park or monitoring network supersedes the LRD boundary.

In addition to adjustments based on new acquisitions or Congressional adjustments to existing boundaries as reflected in the LRD dataset, I&M operational boundaries may include local boundary adjustments or boundaries of affiliated areas near or adjacent to park units. Local boundary adjustments are areas used in monitoring efforts where those areas differ from official administrative boundaries, while affiliated areas are areas owned by agencies other than NPS and are either co-managed by or of some other interest to NPS.

Methods

Official boundary data produced by NPS LRD are combined with alternate boundary datasets managed by I&M staff to produce adjusted NPS unit boundaries. Then, from the adjusted NPS unit boundaries, a subset of parks are extracted creating I&M operational boundaries that are used to support inventory and monitoring activities (Figure 1). I&M operational boundaries are updated with each release of LRD official NPS Boundaries, which occurs quarterly to annually.

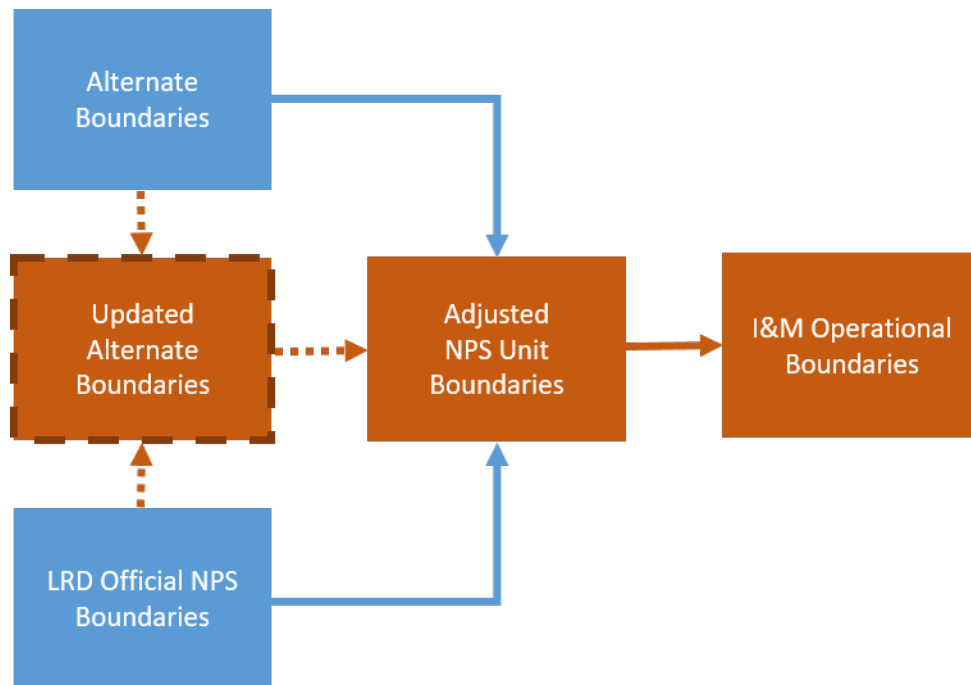


Figure 1. Generalized process flow for boundary processing. Orange boxes indicate output datasets documented by this report. Dash-outlined items may or may not be generated each time data are processed.

This report documents the incorporation of official boundary updates that occurred June 30, 2018 through September 30, 2018 (NPS LRD 2018) into output datasets Adjusted NPS Unit Boundaries (NPS IMD 2018b) and I&M Operational Boundaries (NPS IMD 2018c). In the September 2018 release of the LRD boundaries, changes to ten official boundaries occurred, which are reflected in the output Adjusted NPS Unit Boundaries and I&M Operational Boundaries datasets. Boundary updates due to changes in the NPS LRD (2018) source are summarized in Table 1.

Table 1. Summary of all updated boundaries in output datasets Adjusted NPS Unit Boundaries, and I&M Operational Boundaries. IMD Create Date represents the date the boundary was created from the input source dataset(s) and may not coincide with Land Resources Division (LRD) boundary release date.

Unit Name	Change Reason from Preceding version	IMD Create Date	Adjusted NPS Unit Boundaries	I&M Operational Boundaries
Fort Union Trading Post National Historic Site	Changed by NPS LRD	20181003	Yes	Yes
Grand Canyon National Park	Changed by NPS LRD	20181003	Yes	Yes
Hot Springs National Park	Changed by NPS LRD	20181003	Yes	Yes
Indiana Dunes National Lakeshore	Changed by NPS LRD	20181003	Yes	Yes
Mesa Verde National Park	Changed by NPS LRD	20181003	Yes	Yes
Padre Island National Seashore	Changed by NPS LRD	20181003	Yes	Yes
Petroglyph National Monument	Changed by NPS LRD	20181003	Yes	Yes
Saratoga National Historic Park	Changed by NPS LRD	20181003	Yes	Yes
Tule Springs Fossil Beds National Monument	Changed by NPS LRD	20181003	Yes	Yes
President William Jefferson Clinton Birthplace Home	Changed by NPS LRD	20181003	Yes	No

Source Data

NPS Boundary Data from Land Resources Division

Boundary data from LRD are the official administrative boundaries for NPS. This LRD dataset is updated up to four times per year. The most recent release of LRD boundaries occurred September 30, 2018, which included updates from June 30, 2018 through September 30, 2018 (NPS LRD 2018).

Alternate Boundaries

Two alternate boundary datasets are used to amend LRD data: Affiliated Areas and Local Boundary Adjustments (NPS IMD 2018a). The Affiliated Areas dataset includes areas where the NPS has management authority or interest or areas for which I&M data may be relevant. The Local Boundary Adjustments dataset captures additions or revisions to official LRD boundaries needed to support management activities and may include non-I&M park units. These additions or revisions are requested by parks or monitoring networks based on local knowledge that may not (yet) be captured in the official park boundary data provided by LRD. If a park or network notifies I&M staff that a

boundary as represented by LRD does not reflect the areas where management activities occur, the alternative boundary they provide is added to the one of the alternate datasets.

The alternate boundary datasets have been collected and amended over time, ad hoc, by the Inventory and Monitoring Division (processing details are available in Kinseth and Nelson 2018a). Within the alternate boundary datasets, boundaries are classified as either ‘active’ or ‘inactive’. Only active alternate boundary features are used when creating adjusted NPS unit boundaries (Table 2).

Table 2. Status of boundaries within the affiliated areas and local boundary adjustments source data sets (NPS IMD 2018a).

Source Dataset	Active	Inactive	Total Features
Alternate Boundary - Affiliated Areas	6	2	8
Alternate Boundary - Local Boundary Adjustments	83	0	83

Updating Alternate Boundaries

The alternate boundary datasets were updated prior to producing new adjusted NPS unit boundaries and I&M operational boundaries by incorporating boundary changes or adding parks missing from the LRD source. Updates to alternate boundaries occur if a) IMD has been notified of new alternate park boundaries, b) I&M park units previously missing from the LRD dataset are included in the source LRD dataset, c) changes to boundaries in the official LRD dataset incorporate adjustments previously-documented in alternate boundary datasets.

Incorporate Boundary Changes from Parks and Networks

Existing parks in the alternate boundary dataset were removed or inactivated if one or more of these criteria were met: 1) a park or monitoring network submitted a new alternate boundary; 2) a park or monitoring network rescinded the request to use an existing alternate boundary; 3) a park or monitoring network submitted an update of an existing alternate boundary that differs from the LRD boundary. No new or amended alternate boundary data were received from park or monitoring network staff for the time period documented in this data report.

Incorporate Boundaries Absent from LRD Boundary Data

Using the I&M park list (NPS 2018), LRD boundaries were checked for any I&M park unit boundary absent from the LRD dataset (Table 3). If the alternate boundary datasets did not already contain the absent I&M boundary, a boundary provided by a monitoring network or park was added to the appropriate alternate boundary dataset. Because all I&M park units absent from the LRD dataset were present in the alternate boundary dataset (Table 3), no additions to the alternate boundary datasets were made due to park absence from the source LRD dataset.

Table 3. I&M park unit boundaries absent from the official Land Resources Division (LRD) administrative boundary dataset as of the September 30, 2018 LRD release (NPS LRD, 2018). Create date is the date the park unit was added to the alternate boundary dataset.

Unit Code	Unit Name	Alternate Boundary Exists	Alternate Boundary Create Date	Alternate Boundary Status
NACE	National Capital Parks – East	Yes	11/05/2015	Active
PARA	Parashant National Monument	Yes	11/19/2015	Active
ROVA	Roosevelt-Vanderbilt Headquarters	Yes	2/24/2016	Active
SEKI	Sequoia and Kings Canyon National Parks	Yes	7/13/2009	Active

Incorporate Existing Alternate Boundaries

Any updated LRD boundary was checked against existing alternate boundaries to determine if alternate boundaries had been incorporated into the official LRD boundaries dataset. An existing alternate boundary may supersede any update done by LRD. Existing alternate boundaries were edited or inactivated if one or more of these criteria were met: 1) the LRD boundary update wholly incorporated the alternate boundary; 2) the LRD boundary update partially incorporated the alternate boundary. No changes to the alternate boundary dataset were necessary as a result of changes to LRD boundaries.

If an updated LRD feature did not meet one or more criteria but differed from the alternate boundary previously provided by a monitoring network or park, the previously-provided network- or park-provided boundary would continue to be used.

Table 4 indicates parks in the existing alternate boundary dataset updated by LRD in the September 30, 2018 release (NPS LRD 2018). The existing alternate boundaries for all updated park units superseded all LRD updates (Table 4).

Table 4. Evaluation criteria summary of existing alternate boundaries. IMD Create date is the date the park unit was added to the alternate boundary dataset.

Unit Code	Unit Name	Date of Official Boundary Edit (by Land Resources Division (LRD))	New LRD Data Incorporates Alternate Boundary	New LRD Data Partially Incorporates Alternate Boundary	Existing Alternate Boundary	IMD Create Date	Alternate Boundary Preferred
CHOH	Chesapeake and Ohio Canal National Historical Park	7/2/2018	No	No	Yes	11/05/2015	Yes

Creation of Adjusted NPS Unit Boundaries

The LRD official administrative boundaries (NPS LRD 2018) and the current version of the alternate boundary datasets (NPS IMD 2018a) were the inputs for this release of the Adjusted NPS Unit Boundaries dataset.

Records from the current LRD dataset and the ‘active’ records from the alternate boundary datasets were combined (Kineth and Nelson 2018b). For any park present in both the LRD dataset and the alternate boundary datasets, the active alternate boundary superseded the LRD boundary. This replacement or addition is tracked in the output Adjusted NPS Boundaries dataset in the ‘feature alteration’ attribute (Table 5). Feature alteration identifies how, if at all, a boundary in the Adjusted NPS Boundaries dataset differs from the respective boundary in the LRD dataset.

Table 5. Feature alteration code descriptions for output datasets. Where referenced in the table, source data for Adjusted NPS Unit Boundaries is Land Resources Division (LRD), and source data for I&M Operational Boundaries is Adjusted NPS Unit Boundaries (Figure 1).

Code	Description
NC	No change to output feature from input source (Default value)
AF	Added feature; feature was not present in input source dataset
RS	Replaced feature; input source feature inappropriate for intended use
MS	Input source features merged to create one distinct output feature
SS	Input source feature split to create two or more distinct output features

The un-projected adjusted NPS unit boundaries polygon (IMD_BND_NPSUnits_wgs_py) and centroid point (IMD_BND_NPSUnits_wgs_pt) datasets each contain 465 records. Forty-six LRD

features were superseded by an alternate boundary, one park was split into two parts, and 41 park- or monitoring network-requested features not present in the LRD source dataset were added from the alternate boundary dataset (Table 6).

Table 6. Feature alteration counts for adjusted NPS unit boundary datasets compared to input Land Resources Division source.

Dataset	No alteration to feature (NC)	Added feature (AF)	Replaced feature (RS)	Merged feature (MS)	Split feature (SS)	Total Features
Adjusted NPS Unit Boundaries	376	41	46	0	2	465

The ‘change reason from preceding version’ attribute in the adjusted NPS unit boundaries datasets indicates why a boundary change occurred (Table 7). Change reason applies only to adjusted NPS unit boundaries (NPS IMD 2018b) when compared to the preceding version of adjusted NPS unit boundaries (NPS IMD 2018d), and describes why, when compared to the preceding version, a feature changed.

Table 7. Counts of change reason from preceding version for adjusted NPS unit boundaries dataset.

Change Reason from Preceding Version	Count
Changed by NPS LRD	10
No change	455
Total	465

Inactivation of Obsolete Adjusted NPS Unit Boundaries

In the adjusted NPS unit boundaries output, if an existing boundary changed, it was assigned a status of ‘inactive’ (Table 8). No boundaries were inactivated during this processing effort.

Table 8. Active and inactive feature counts for adjusted NPS unit boundaries dataset.

Status	Count
Active	465
Inactive	0
Total	465

Creation of I&M Operational Boundaries

Active adjusted NPS unit boundary records were filtered to include only I&M parks (NPS 2018). For some parks, boundaries were merged or replaced. As with the adjusted NPS unit boundaries datasets, these changes are tracked in the I&M Operational Boundaries dataset in the feature alteration attribute (Table 5). Feature alteration identifies how, if at all, a feature in the I&M Operational Boundaries dataset differs from the respective feature in the Adjusted NPS Boundaries source dataset.

The output I&M operational boundaries database (IMD_BND_Units.gdb) contains four datasets, each representing 293 boundaries: un-projected polygons (IMD_BND_Units_wgs_py), un-projected centroid points (IMD_BND_Units_wgs_pt), projected polygons (IMD_BND_Units_albers_py), and un-projected services polygons (IMD_BND_UnitsService_wgs_py). In the output datasets, 18 adjusted NPS unit boundaries for park units with differing designations which are managed as one unit, such as Denali National Park and Denali National Preserve, were merged to create nine I&M operational boundaries (Table 9).

The un-projected service polygon dataset (IMD_BND_UnitsService_wgs_py) was created explicitly for use in visualization of IMD boundaries in web map applications. For this dataset only, the highly complex boundary for Appalachian National Scenic Trail (APPA) was replaced with a de-densified version, producing a simplified APPA boundary that can be rendered quickly in web maps. This replacement of the APPA boundary is tracked in the feature alteration attribute (Table 9).

Table 9. Feature alteration counts for I&M operational boundaries datasets compared to input adjusted NPS unit boundary source dataset.

Dataset	No alteration to feature (NC)	Added feature (AF)	Replaced feature (RS)	Merged feature (MS)	Split feature (SS)
I&M Operational Boundaries	284	0	0	9	0
I&M Operational Boundaries for Map Services	283	0	1	9	0

The ‘change reason from preceding version’ value in the I&M operational boundaries dataset was inherited from the adjusted NPS unit boundary records (Table 10). Change reason applies to I&M operational boundaries (NPS IMD 2018b) when compared to the preceding version of I&M operational boundaries (NPS IMD 2018e) and describes why, when compared to the preceding version, a feature changed.

Table 10. Counts of change reason from preceding version for I&M operational boundaries dataset.

Change Reason from Preceding Version	Count
Changed by NPS LRD	9
No change	284
Total	293

Inactivation of Obsolete I&M Operational Boundaries

In the I&M operational boundaries output, if an existing park boundary changed, it was assigned a status of ‘inactive’ (Table 11). No boundaries were inactivated during this processing effort.

Table 11. Active and inactive feature counts for I&M operational boundaries dataset.

Status	Count
Active	293
Inactive	0
Total	293

Additional Processing

To create Adjusted NPS Unit Boundaries and I&M Operational Boundaries outputs, deviations from the published SOPs were required. To format data attributes consistently across the output datasets through the application of database domains, the script(s) noted in Appendices D and E were run after all other processing was complete.

Code Availability

See Appendices A - E for the Python toolbox scripts and tools used to execute these methods. Appendix F documents the software tools used to create the output datasets (Table 12).

Data Records

Two outputs (in file geodatabase database format) were generated as a part of this effort (Table 12):

1. **NPS Adjusted Unit Boundaries (IMD_BND_NPSUnits.gdb).** This database represents 465 adjusted NPS unit boundaries and contains two datasets which are geographically referenced to the WGS84 datum and not spatially projected (WKID 4326): Database domains are represented by seven tables.
 - a. Adjusted NPS Unit Boundary. Un-projected polygon features (IMD_BND_NPSUnits_wgs_py) representing 465 adjusted NPS unit boundaries.
 - b. Adjusted NPS Unit Boundary. Un-projected point features (IMD_BND_NPSUnits_wgs_pt) representing the centroid locations of 465 adjusted NPS unit boundaries.

Available at <https://irma.nps.gov/DataStore/Reference/Profile/2266196>

2. **I&M Operational Boundaries (IMD_BND_Units.gdb).** This database represents 293 I&M operational boundaries and contains four datasets: Database domains are represented by seven tables.
 - a. I&M Operational Boundaries. Un-projected polygon features (IMD_BND_Units_wgs_py) representing 293 I&M operational boundaries. This feature class is geographically referenced to the WGS84 datum and is not spatially projected (WKID 4326).
 - b. I&M Operational Boundaries. Un-projected services polygon features intended for use in web map applications (IMD_BND_UnitsService_wgs_py) representing 293 I&M operational boundaries including a simplified Appalachian National Scenic Trail boundary. This feature class is geographically referenced to the WGS84 datum and is not spatially projected (WKID 4326).
 - c. I&M Operational Boundaries. Projected polygon features (IMD_BND_Units_albers_py) representing 293 I&M operational boundaries. This feature class is spatially projected to the USA Contiguous Albers Equal Area Conic USGS spatial reference (WKID 102003).
 - d. I&M Operational Boundaries. Un-projected point features (IMD_BND_Units_wgs_pt) representing the centroid of 293 I&M operational boundary polygon features (IMD_BND_Units_wgs_py). This feature class is geographically referenced to the WGS84 datum and not spatially projected (WKID 4326).

Available at <https://irma.nps.gov/DataStore/Reference/Profile/2266200>

Table 12. Overview of datasets and their provenance. Temporal currency of data sets is based on the date of publication of the Land Resources Division source data used.

Source Data	Resultant Data	Number of Records	Temporal Currency
LRD Official Boundaries (NPS LRD 2018) Alternate Boundaries (NPS IMD 2018a)	Adjusted NPS Unit Boundaries	465	2018-09-30
Adjusted NPS Unit Boundaries	I&M Operational Boundaries	293	2018-09-30

Data Quality Evaluation

Spatial and attribute quality is not provided in the official boundary data provided by the NPS Land Resources Division (NPS LRD 2018). The use of alternate boundaries based on local expertise to generate adjusted and I&M operational boundaries is considered to be a quality control process whereby official boundaries are corrected to represent park boundaries as of 9/30/2018. The output data sets are considered suitable for use to meet the needs of the NPS Inventory & Monitoring Program.

Usage Notes

All output datasets are intended and suitable for use in the collection, analysis, and reporting of inventory and monitoring data collected by NPS Inventory and Monitoring Division. NPS Unit Boundaries are intended for use in inventories (DeVivo 2019) and other related work. I&M Operational Boundaries are intended for use when deriving areas of analysis for—and boundary-based statistics about—park units.

With the exception of boundaries from the dataset published by the NPS Land Resources Division (NPS LRD 2018), park boundaries represented in this report are not official.

References

- DeVivo, J. C., L. Nelson, M. Kineth, T. Philippi, and W. B. Monahan. 2018. Protocol for monitoring environmental setting of National Park Service units: landscape dynamics, climate, and hydrology. Natural Resource Report NPS/IMD/NRR—2018/1844. National Park Service, Fort Collins, Colorado.
- DeVivo, J. C. 2019. Inventories 2.0: A plan for the next generation of NPS natural resource inventories. Natural Resource Report NPS/NRSS/NRR—2019/2007. National Park Service, Fort Collins, Colorado.
- Kineth, M. and L. Nelson. 2018a. NPS unit boundary processing: Park operational administrative boundaries. Inventory & Monitoring Division Standard Operating Procedure NPS/IMD/SOP—2.11.2. National Park Service, Fort Collins, Colorado.
<https://irma.nps.gov/DataStore/Reference/Profile/2256634>
- Kineth, M. and L. Nelson. 2018b. NPS unit boundary processing: Reconciliation of official Lands Division and park operational administrative boundaries. Inventory & Monitoring Division Standard Operating Procedure NPS/IMD/SOP—2.11.3. National Park Service, Fort Collins, Colorado.
<https://irma.nps.gov/DataStore/Reference/Profile/2256638>
- Nelson, L. and M. Kineth. 2018c. NPS unit boundary processing: Boundaries for use for analysis and reporting by the Inventory & Monitoring Division. Inventory & Monitoring Program Standard Operating Procedure NPS/NRSS/IMD/SOP-2.11.4. National Park Service, Fort Collins, Colorado.
<https://irma.nps.gov/DataStore/Reference/Profile/2256648>
- National Park Service Inventory & Monitoring Division (NPS IMD). 2018a. Unit Boundary Processing – Affiliated Areas and Local Boundary Adjustments. *National Park Service Data Store* <https://irma.nps.gov/DataStore/Reference/Profile/2256634>
- National Park Service Inventory & Monitoring Division (NPS IMD). 2018b. Adjusted NPS Unit Boundaries (Unofficial), Fall 2018. *National Park Service Data Store* <https://irma.nps.gov/DataStore/Reference/Profile/2266196>
- National Park Service Inventory & Monitoring Division (NPS IMD). 2018c. I&M Operational Boundaries – NPS Inventory and Monitoring (I&M) Boundaries, Fall 2018. *National Park Service Data Store* <https://irma.nps.gov/DataStore/Reference/Profile/2266200>
- National Park Service Inventory & Monitoring Division (NPS IMD). 2018d. Unit Boundary Processing – NPS Unit Boundaries (Unofficial). *National Park Service Data Store* <https://irma.nps.gov/DataStore/Reference/Profile/2253978>
- National Park Service Inventory & Monitoring Division (NPS IMD). 2018e. I&M Park Boundaries. *National Park Service Data Store*. <https://irma.nps.gov/DataStore/Reference/Profile/2253985>

National Park Service Inventory & Monitoring Division (NPS IMD). 2018f. Geodatabase domain tables used in Inventory and Monitoring Division unit boundary processing. *National Park Service Data Store* <https://irma.nps.gov/DataStore/Reference/Profile/2267506>

National Park Service (NPS). 2018. IRMA Unit Database and visualization feature update. *National Park Service Data Store* <https://irma.nps.gov/DataStore/Reference/Profile/2256640>

National Park Service Land Resources Division (NPS LRD). 2018. Administrative Boundaries of National Park System Units 9/30/2018 - National Geospatial Data Asset (NGDA) NPS National Parks Dataset. NPS - Land Resources Division. *National Park Service Data Store* <https://irma.nps.gov/DataStore/Reference/Profile/2225713>

Appendix A. Python Toolbox Code Affiliated Areas and Local Boundary Adjustments Verification

The following includes python toolbox code for two tools referenced in this report: **Affiliated Areas and Local Boundary Adjustments Verification**.

```
import arcpy
import os, time
from datetime import datetime

class Toolbox(object):
    def __init__(self):
        """Define the toolbox (the name of the toolbox is the name
of the
        .pyt file)."""
        self.label = "AlternateUnitBounds_Processing"
        self.alias = ""

        # List of tool classes associated with this toolbox
        self.tools = [FindMissingBounds, FindUpdatedIMDParks]

class FindMissingBounds(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)."""
        self.label = "Find Bounds Missing from Lands"
        self.description = "Create tbl_MissingFromLands table in
UnitBounds.gdb"
        self.canRunInBackground = False

    def getParameterInfo(self):
        """Define parameter definitions"""

        param0 = arcpy.Parameter(
            displayName = "Source NPS Lands Shapefile",
            name = "npsAdminBounds",
            datatype = "GPFeatureLayer",
            parameterType = "Required",
            direction = "Input")
        param0.value =
r"X:\ProjectData\Data_Processing\Bounds_Processing\NPS_Boundary_Sou
rce\nps_boundary\nps_boundary.shp"

        param1 = arcpy.Parameter(
```

```

        displayName = "Output GeoDatabase",
        name = "outDB",
        datatype = "DEWorkspace",
        parameterType = "Required",
        direction = "Input")

    param2 = arcpy.Parameter(
        displayName = "IMD Bounds Lookup Table",
        name = "altBounds",
        datatype = "DETable",
        parameterType = "Required",
        direction = "Input")
    #param2.value =

    params = [param0, param1, param2]
    return params

def isLicensed(self):
    """Set whether tool is licensed to execute."""
    return True

def updateParameters(self, parameters):
    """Modify the values and properties of parameters before
internal
    validation is performed. This method is called whenever a
parameter
    has been changed."""
    return

def updateMessages(self, parameters):
    """Modify the messages created by internal validation for
each tool
    parameter. This method is called after internal
validation."""
    return

def execute(self, parameters, messages):
    message = ""
    today=datetime.now()
    datestamp = str(today.isoformat()).replace('-', '')[0:8]

    arcpy.env.overwriteOutput = 1
    outputMissingTableName = "tbl_MissingFromLands"

```

```

tempLayer = "tempLayer"
lookupTable = "lookupTable"
outputMissingTable = os.path.join(parameters[1].valueAsText,
outputMissingTableName + '_' + datestamp)

# Join lookup to nps_boundary.shp and copy missing records
to tbl_MissingFromLands
    arcpy.MakeFeatureLayer_management(parameters[0].valueAsText,
tempLayer); messages.addGPMessages()
    arcpy.MakeTableView_management(parameters[2].valueAsText,
lookupTable); messages.addGPMessages()
    arcpy.AddJoin_management(lookupTable, "Code", tempLayer ,
"UNIT_CODE"); messages.addGPMessages() #"Unit_Code"
    arcpy.SelectLayerByAttribute_management(lookupTable,
"NEW_SELECTION", "nps_boundary.UNIT_CODE IS NULL");
messages.addGPMessages()

    arcpy.CopyRows_management(lookupTable, outputMissingTable);
messages.addGPMessages()
    arcpy.RemoveJoin_management(lookupTable);
messages.addGPMessages()

    itemsToDelete = [tempLayer, lookupTable]
    for item in itemsToDelete:
        if arcpy.Exists(item):
            arcpy.Delete_management(item);
messages.addGPMessages()

class FindUpdatedIMDParks(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)."""
        self.label = "Find Updated IMD Parks"
        self.description = "Create tbl_UpdatedIMDParks table in
UnitBounds.gdb"
        self.canRunInBackground = False

    def getParameterInfo(self):
        """Define parameter definitions"""

        param0 = arcpy.Parameter(
            displayName = "Source NPS Lands Shapefile",
            name = "npsAdminBounds",
            datatype = "GPFeatureLayer",

```

```

        parameterType = "Required",
        direction = "Input")
    param0.value =

    param1 = arcpy.Parameter(
        displayName = "Output GeoDatabase",
        name = "outDB",
        datatype = "DEWorkspace",
        parameterType = "Required",
        direction = "Input")

    param2 = arcpy.Parameter(
        displayName = "IMD Bounds Lookup Table",
        name = "altBounds",
        datatype = "DETable",
        parameterType = "Required",
        direction = "Input")
    #param2.value =

    param3 = arcpy.Parameter(
        displayName = "Date to Check",
        name = "dateCheck",
        datatype = "GPDate",
        parameterType = "Required",
        direction = "Input")

    params = [param0, param1, param2, param3]
    return params

def isLicensed(self):
    """Set whether tool is licensed to execute."""
    return True

def updateParameters(self, parameters):
    """Modify the values and properties of parameters before
internal
    validation is performed. This method is called whenever a
parameter
    has been changed."""
    return

def updateMessages(self, parameters):

```

```

        """Modify the messages created by internal validation for
each tool
        parameter. This method is called after internal
validation."""
        return

def execute(self, parameters, messages):
    message = ""
    today=datetime.now()
    datestamp = str(today.isoformat()).replace('-', '')[0:8]

    arcpy.env.overwriteOutput = 1
    outputUpdatedTableName = "tbl_IMDParksUpdated"
    tempLayer = "tempLayer"
    lookupTable = "lookupTable"
    outputUpdatedTable = os.path.join(parameters[1].valueAsText,
outputUpdatedTableName + '_' + datestamp)
    checkDate = "date '" + parameters[3].value.strftime('%Y-%m-%d') + "'"
    arcpy.AddMessage("\ncheckDate = " + checkDate)
    selectQuery = '"nps_boundary.DATE_EDIT" >= %s' % checkDate
    arcpy.AddMessage("\nselectQuery = " + selectQuery)

    # Join lookup to nps_boundary.shp and copy missing records
to tbl_MissingFromLands
    arcpy.MakeFeatureLayer_management(parameters[0].valueAsText,
tempLayer); messages.addGPMessages()
    arcpy.MakeTableView_management(parameters[2].valueAsText,
lookupTable); messages.addGPMessages()
    arcpy.AddJoin_management(lookupTable, "Code", tempLayer ,
"UNIT_CODE"); messages.addGPMessages() #"Unit_Code"
    arcpy.SelectLayerByAttribute_management(lookupTable,
"NEW_SELECTION", "nps_boundary.UNIT_CODE IS NOT NULL");
messages.addGPMessages()
    arcpy.SelectLayerByAttribute_management(lookupTable,
"SUBSET_SELECTION", selectQuery); messages.addGPMessages()

    arcpy.CopyRows_management(lookupTable, outputUpdatedTable);
messages.addGPMessages()
    arcpy.RemoveJoin_management(lookupTable);
messages.addGPMessages()

    itemsToDelete = [tempLayer, lookupTable]

```

```
    for item in itemsToDelete:
        if arcpy.Exists(item):
            arcpy.Delete_management(item);
messages.addGPMessages()
```

Appendix B. Python Toolbox Code Processing NPS Unit Bounds

The following includes python toolbox code for processing and creating IMD_BND_NPSUnits.gdb.

```
import arcpy
import os, time
from datetime import datetime

class Toolbox(object):
    def __init__(self):
        """Define the toolbox (the name of the toolbox is the name
of the
        .pyt file)."""
        self.label = "UnitBounds_Processing"
        self.alias = ""

        # List of tool classes associated with this toolbox
        self.tools = [UpdateUnitBounds]

class UpdateUnitBounds(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)."""
        self.label = "Update Unit Bounds"
        self.description = "Create or update
IMD_BND_NPSUnits_wgs_py feature class in IMD_BND_NPSUnits.gdb"
        self.canRunInBackground = False

    def getParameterInfo(self):
        """Define parameter definitions"""

        param0 = arcpy.Parameter(
            displayName = "Source NPS Lands Shapefile",
            name = "npsAdminBounds",
            datatype = "GPFeatureLayer",
            parameterType = "Required",
            direction = "Input")
        param0.value =
r"X:\ProjectData\Data_Processing\Bounds_Processing\NPS_Boundary_Sou
rce\nps_boundary\nps_boundary.shp"

        param1 = arcpy.Parameter(
            displayName = "Output GeoDatabase",
```

```

        name = "outDB",
        datatype = "DEWorkspace",
        parameterType = "Required",
        direction = "Input")

    param2 = arcpy.Parameter(
        displayName = "Alternate Bounds feature class",
        name = "altBounds",
        datatype = "GPFeatureLayer",
        parameterType = "Required",
        direction = "Input")
    param2.value =
r"X:\ProjectData\Data_Processing\Bounds_Processing\Data\UnitBound_P
rocessing.gdb\AdjustmentsLocalBoundary" #AlternateBounds"

    param3 = arcpy.Parameter(
        displayName = "Affiliated Areas feature class",
        name = "affBounds",
        datatype = "GPFeatureLayer",
        parameterType = "Required",
        direction = "Input")
    param3.value =
r"X:\ProjectData\Data_Processing\Bounds_Processing\Data\UnitBound_P
rocessing.gdb\AffiliatedAreas"
#"D:\Workspace\UnitBound_Processing.gdb\AffiliatedAreas"

    param4 = arcpy.Parameter(
        displayName = "Folder Containing Metadata Templates",
        name = "metaFolder",
        datatype = "Folder",
        parameterType = "Required",
        direction = "Input")
    param4.value =
r"X:\ProjectData\Data_Processing\Bounds_Processing\Metadata_Templat
es"

    params = [param0, param1, param2, param3, param4] #param5
    return params

def isLicensed(self):
    """Set whether tool is licensed to execute."""
    return True

```

```

    def updateParameters(self, parameters):
        """Modify the values and properties of parameters before
internal
validation is performed. This method is called whenever a
parameter
has been changed."""
        return

    def updateMessages(self, parameters):
        """Modify the messages created by internal validation for
each tool
parameter. This method is called after internal
validation."""
        return

    def deleteUnit(self, targetLayer, codeSnippet):
        """
        Deletes selected feature from target layer
        """

arcpy.SelectLayerByAttribute_management(targetLayer,"NEW_SELECTION"
, codeSnippet)
    arcpy.DeleteFeatures_management(targetLayer)
    arcpy.AddMessage('\n Deleted ' + codeSnippet)

arcpy.SelectLayerByAttribute_management(targetLayer,"CLEAR_SELECTIO
N")

    def calcField(self, targetLayer, targetField, selectSnippet,
expr, switch=None, targetField2=None, selectSnippet2=None,
expr2=None):
        """
        Calculates field for selected records
        """

arcpy.SelectLayerByAttribute_management(targetLayer,"NEW_SELECTION"
, selectSnippet)
        if switch:

arcpy.SelectLayerByAttribute_management(targetLayer,"SWITCH_SELECTI
ON", selectSnippet)

```

```

        arcpy.CalculateField_management(targetLayer, targetField,
expr, "PYTHON")
        arcpy.AddMessage('\nCalculated Field: ' + selectSnippet +
' as ' + expr)

arcpy.SelectLayerByAttribute_management(targetLayer,"CLEAR_SELECTIO
N")

        if targetField2:

arcpy.SelectLayerByAttribute_management(targetLayer,"NEW_SELECTION"
, selectSnippet)
            arcpy.CalculateField_management(targetLayer,
targetField2, expr2, "PYTHON")

arcpy.SelectLayerByAttribute_management(targetLayer,"CLEAR_SELECTIO
N")
            arcpy.AddMessage('\nCalculated Field: ' +
selectSnippet2 + ' as ' + expr2)

def execute(self, parameters, messages):
    message = ""
    today=datetime.now()
    datestamp = str(today.isoformat()).replace('-',')[0:8]
    metadataTemplate =
['ES_UnitBoundsTemplate_FDGC.xml','ES_UnitBoundsPtTemplate_FDGC.xml
']

    arcpy.env.overwriteOutput = 1
    # Lands bounds have M and Z values for some reason
    arcpy.env.outputMFlag = "Disabled"
    arcpy.env.outputZFlag = "Disabled"
    outputLandsFeatureClassName = "nps_boundary"
    outputGDBFeatureClassName = "IMD_BND_NPSUnits_wgs_py"
    outputGDBFeatureClassNamePt = "IMD_BND_NPSUnits_wgs_pt"
    tempDissolveFeatureClassName = "lands_dissolve"
    tempMergedFeatureClassName = "lands_merged"
    tempLayer = "tempLayer"
    altLayer = "altLayer"
    affLayer = "affLayer"
    mergeLayer = "mergeLayer"
    outputLandsFeatureClass =
os.path.join(parameters[1].valueAsText,
outputLandsFeatureClassName)

```

```

        outputGDBFeatureClass =
os.path.join(parameters[1].valueAsText, outputGDBFeatureClassName)
        outputGDBFeatureClassPt =
os.path.join(parameters[1].valueAsText,
outputGDBFeatureClassNamePt)
        tempDissolveFeatureClass =
os.path.join(parameters[1].valueAsText,
tempDissolveFeatureClassName)
        tempMergedFeatureClass =
os.path.join(parameters[1].valueAsText, tempMergedFeatureClassName)
        altFeatureClass = parameters[2].valueAsText
        affFeatureClass = parameters[3].valueAsText
        outSR =
'GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID["WGS_1984",63781
37,298.257223563]],PRIMEM["Greenwich",0],UNIT["Degree",0.0174532925
19943295]]'

```

```

unitsToDelete = [
    'UNIT_CODE = \'ACAD\'',
    'UNIT_CODE = \'ALKA\'',
    'UNIT_CODE = \'AMME\'',
    'UNIT_CODE = \'ANTI\'',
    'UNIT_CODE = \'ARHO\'',
    'UNIT_CODE = \'BICA\'',
    'UNIT_CODE = \'BRCR\'',
    'UNIT_CODE = \'CAHE\'',
    'UNIT_CODE = \'CATO\'',
    'UNIT_CODE = \'CAWO\'',
    'UNIT_CODE = \'CHOH\'',
    'UNIT_CODE = \'CONG\'',
    'UNIT_CODE = \'CRLA\'',
    'UNIT_CODE = \'DRTO\'',
    'UNIT_CODE = \'FRDO\'',
    'UNIT_CODE = \'GREE\'',
    'UNIT_CODE = \'GLAC\'',
    'UNIT_CODE = \'GWMP\'',
    'UNIT_CODE = \'GRSM\'',
    'UNIT_CODE = \'HAFE\'',
    'UNIT_CODE = \'HAVO\'',
    'UNIT_CODE = \'INPE\'',
    'UNIT_CODE = \'KIMO\'',
    'UNIT_CODE = \'KLGO\'',
    'UNIT_CODE = \'LACH\'',

```

```

'UNIT_CODE = \'LAKE\'',
'UNIT_CODE = \'LAVO\'',
'UNIT_CODE = \'MABE\'',
'UNIT_CODE = \'MABI\'',
'UNIT_CODE = \'MANA\'',
'UNIT_CODE = \'MONO\'',
'UNIT_CODE = \'NACE\'',
'UNIT_CODE = \'NATR\'',
'UNIT_CODE = \'NISI\'',
'UNIT_CODE = \'NPSA\'',
'UNIT_CODE = \'PISC\'',
'UNIT_CODE = \'OLYM\'',
'UNIT_CODE = \'ORCA\'',
'UNIT_CODE = \'PIPE\'',
'UNIT_CODE = \'PRWI\'',
'UNIT_CODE = \'PUHO\'',
'UNIT_CODE = \'REDW\'',
'UNIT_CODE = \'ROCR\'',
'UNIT_CODE = \'SEBE\'',
'UNIT_CODE = \'TUPE\'',
'UNIT_CODE = \'VALR\'',
'UNIT_CODE = \'WAPA\'',
'UNIT_CODE = \'WOTR\']

```

```

# #####
# if spatial ref NOT WKID/GCSCode 4269 (NAD83 with
Authority), project param0 TO WKID 4269; else pass
bds = parameters[0].valueAsText
bds_sr = arcpy.Describe(bds).spatialReference
nad83 = arcpy.SpatialReference(4269)
nad83_11 = arcpy.SpatialReference(6318)
transform_11 = 'WGS_1984_(ITRF00)_To_NAD_1983 +
WGS_1984_(ITRF00)_To_NAD_1983_2011'
bds_prj = os.path.join(parameters[1].valueAsText, "TEMP1")
bds_prj1 = os.path.join(parameters[1].valueAsText, "TEMP2")
if bds_sr.GCSCode == 6318:
    pass
else:
    arcpy.Project_management(bds,bds_prj,nad83);
messages.addGPMessages()
    arcpy.RepairGeometry_management(bds_prj);
messages.addGPMessages()

```

```

arcpy.Project_management(bds_prj,bds_prj1,nad83_11,transform_11);
messages.addGPMessages()
    arcpy.RepairGeometry_management(bds_prj1);
messages.addGPMessages()
    arcpy.Delete_management(bds_prj);
messages.addGPMessages()
    bds = bds_prj1

# #####

    # Dissolve nps_boundary.shp to remove lingering single-part
features
    arcpy.CopyFeatures_management(bds,
outputLandsFeatureClass); messages.addGPMessages()
#parameters[0].valueAsText
    arcpy.Dissolve_management(bds, tempDissolveFeatureClass,
"UNIT_CODE;UNIT_NAME;UNIT_TYPE;DATE_EDIT;GIS_NOTES", "#",
"MULTI_PART"); messages.addGPMessages() #parameters[0].valueAsText
    arcpy.RepairGeometry_management(tempDissolveFeatureClass);
messages.addGPMessages()
    arcpy.Delete_management(bds_prj1); messages.addGPMessages()

arcpy.AddField_management(tempDissolveFeatureClass,"Unit_Code_Displ
ay","TEXT","","","25","","","NULLABLE","NON_REQUIRED");
messages.addGPMessages()

    # Make changes to LRD versions of CRMO/CRMP, create CRMM
tLayer = 'tLayer'
    arcpy.MakeFeatureLayer_management(tempDissolveFeatureClass,
tLayer); messages.addGPMessages()

arcpy.SelectLayerByAttribute_management(tLayer,"NEW_SELECTION","UNI
T_CODE = 'CRMO'"); messages.addGPMessages()

arcpy.CalculateField_management(tLayer,"UNIT_CODE",' "CRMG" ');
messages.addGPMessages()

arcpy.SelectLayerByAttribute_management(tLayer,"ADD_TO_SELECTION","
UNIT_CODE = 'CRMP'"); messages.addGPMessages()

arcpy.CopyFeatures_management(tLayer,"in_memory\CRMM_temp");
messages.addGPMessages()

```

```
arcpy.Dissolve_management("in_memory\CRMM_temp","in_memory\CRMM","DATE_EDIT"); messages.addGPMessages()
```

```
arcpy.JoinField_management("in_memory\CRMM","DATE_EDIT","in_memory\CRMM_temp","DATE_EDIT",["UNIT_CODE","GIS_Notes","UNIT_NAME","PARKNAME","STATE","REGION","GNIS_ID","UNIT_TYPE","CREATED_BY","METADATA"])
```

```
arcpy.CalculateField_management("in_memory\CRMM","UNIT_CODE","CRMM"); messages.addGPMessages()
```

```
arcpy.CalculateField_management("in_memory\CRMM","UNIT_NAME","Craters of the Moon National Monument and Preserve"); messages.addGPMessages()
```

```
arcpy.SelectLayerByAttribute_management(tLayer,"CLEAR_SELECTION"); messages.addGPMessages()
    arcpy.Append_management("in_memory\CRMM",tLayer,"NO_TEST"); messages.addGPMessages()
```

```
# #####
```

```
# Prepare feature layer for use in selection and delete features replaced by AlternateBounds features.
```

```
arcpy.MakeFeatureLayer_management(tLayer, tempLayer); messages.addGPMessages()
arcpy.AddMessage('\n' + str(unitsToDelete) + '\n')
for item, stmt in enumerate(unitsToDelete):
    UpdateUnitBounds.deleteUnit(self, tempLayer, stmt)
    #deleteUnit(self, tempLayer, stmt)
```

```
#arcpy.RepairGeometry_management(tempLayer); messages.addGPMessages()
```

```
# Update Feature_Source and Source_Metadata attributes
```

```
arcpy.AddField_management(tempLayer,"Feature_Source","TEXT","#","#",80,"#","NULLABLE", "#", "Feature_Source"); messages.addGPMessages()
```

```
arcpy.AddField_management(tempLayer,"Source_Metadata","TEXT","#","#",100,"#","NULLABLE", "#"); messages.addGPMessages()
```

```

arcpy.AddField_management(tempLayer,"Source_Details","TEXT","#","#",
,120,"#","NULLABLE", "#"); messages.addGPMessages()

    UpdateUnitBounds.calcField(self, tempLayer,
"Feature_Source", 'GIS_Notes LIKE \'LEGACY%\'', '"Legacy Sources"')
    UpdateUnitBounds.calcField(self, tempLayer,
"Feature_Source", 'GIS_Notes LIKE \'Lands%\'', '"NPS Land Resources
Division"', None, "Source_Metadata", 'GIS_Notes LIKE \'Lands%\'',
'"https://irma.nps.gov/DataStore/Reference/Profile/2194483?lnv=true
"')

    UpdateUnitBounds.calcField(self, tempLayer,
"Feature_Source", 'GIS_Notes LIKE \'Lands%\' OR GIS_Notes LIKE
\'LEGACY%\' OR UNIT_CODE NOT LIKE \'CRMM%\'', '"NPS Inventory and
Monitoring Division - Central Office"', 1)
    arcpy.CalculateField_management(tempLayer,
"Source_Details", '!GIS_Notes!', 'PYTHON_9.3');
messages.addGPMessages()

    # Prepare feature layers for alternate bounds and
    affiliated areas
    arcpy.MakeFeatureLayer_management(altFeatureClass,
altLayer); messages.addGPMessages()

arcpy.SelectLayerByAttribute_management(altLayer,"NEW_SELECTION",
"Lifecycle = 'Active'"); messages.addGPMessages()
    arcpy.MakeFeatureLayer_management(affFeatureClass,
affLayer); messages.addGPMessages()

arcpy.SelectLayerByAttribute_management(affLayer,"NEW_SELECTION",
"Lifecycle = 'Active'"); messages.addGPMessages()

    # Merge in alternate bounds and affiliated areas, Remove
    extraneous fields
    arcpy.Merge_management([tempLayer, altLayer, affLayer],
tempMergedFeatureClass); messages.addGPMessages()
    arcpy.RepairGeometry_management(tempMergedFeatureClass);
messages.addGPMessages()

arcpy.DeleteField_management(tempMergedFeatureClass,["STATE","REGIO
N","CREATED_BY","METADATA","PARKNAME","Full_Name","UNIT_TYPE"]);
messages.addGPMessages()

```

```

arcpy.AddField_management(tempMergedFeatureClass,"Full_Name","TEXT",
"#", "#", 100, "#", "NULLABLE", "#"); messages.addGPMessages()
    expression = '!UNIT_NAME!'
    arcpy.CalculateField_management(tempMergedFeatureClass,
"Full_Name", expression, "PYTHON"); messages.addGPMessages()

arcpy.AddField_management(tempMergedFeatureClass,"RedactReason","TEXT",
"#", "#", 25, "#", "NULLABLE", "#"); messages.addGPMessages()

arcpy.AddField_management(tempMergedFeatureClass,"DSLLink","TEXT", "#",
"#", 200, "#", "NULLABLE", "#"); messages.addGPMessages()

arcpy.CalculateField_management(tempMergedFeatureClass,"DSLLink",'https://irma.nps.gov/DataStore/Reference/Profile/2245889?lnv=True')
; messages.addGPMessages()

arcpy.AddField_management(tempMergedFeatureClass,"ChangeReason","TEXT",
"", "", 50, "", "NULLABLE", "NON_REQUIRED", "");
messages.addGPMessages()

    DomainTable =
r"E:\Projects_Workspace\Bounds_Processing_Workspace\Data_forTesting\test\ReferenceTables.gdb\ChangeReason"
    DomainTable2 =
r"E:\Projects_Workspace\Bounds_Processing_Workspace\Data_forTesting\test\ReferenceTables.gdb\RedactReason"

arcpy.AssignDomainToField_management(tempMergedFeatureClass,"ChangeReason","ChangeReason_Domain"); messages.addGPMessages()

arcpy.AssignDomainToField_management(tempMergedFeatureClass,"RedactReason","RedactReason_Domain"); messages.addGPMessages()

arcpy.CalculateField_management(tempMergedFeatureClass,"ChangeReason",'None'); messages.addGPMessages()

arcpy.CalculateField_management(tempMergedFeatureClass,"RedactReason",'None'); messages.addGPMessages()

arcpy.CalculateField_management(tempMergedFeatureClass,"DataCertificationLevel",'Accepted'); messages.addGPMessages()

```

```

arcpy.CalculateField_management(tempMergedFeatureClass,"FeatureQual
ifier","NC"); messages.addGPMessages() #ProcessingQualifier

arcpy.CalculateField_management(tempMergedFeatureClass,"UnitCodeQual
ifier","SC"); messages.addGPMessages()

    # Update Feature_Source and Source_Metadata attributes for
merged features
    arcpy.MakeFeatureLayer_management(tempMergedFeatureClass,
mergeLayer); messages.addGPMessages()
    UpdateUnitBounds.calcField(self, mergeLayer,
"Feature_Source", 'Source_Details LIKE \'Good%\'', '"Legacy
Sources"')
    UpdateUnitBounds.calcField(self, mergeLayer,
"Feature_Source", 'Feature_Source IS Null', '"NPS Inventory and
Monitoring Division - Central Office"')
    UpdateUnitBounds.calcField(self, mergeLayer, "CreateDate",
'CreateDate LIKE \'2%\'', '""'+datestamp+"", 1)

arcpy.CalculateField_management(mergeLayer,"Unit_Code_Display","!UN
IT_CODE!", "PYTHON_9.3"); messages.addGPMessages()
    UpdateUnitBounds.calcField(self, mergeLayer, "Lifecycle",
'GIS_Notes LIKE \'Lands%\'' OR GIS_Notes LIKE \'LEGACY%\'',
'"Active"')
    arcpy.DeleteField_management(mergeLayer,["GIS_Notes"]);
messages.addGPMessages()

    if arcpy.Exists(outputGDBFeatureClassPt):
arcpy.Delete_management(outputGDBFeatureClassPt);
messages.addGPMessages()

    # Version existing feature class; Project new feature class
and import metadata
    if arcpy.Exists(outputGDBFeatureClass):
        arcpy.Rename_management(outputGDBFeatureClass,
outputGDBFeatureClass + '_' + datestamp); messages.addGPMessages()

        transform = 'WGS_1984_(ITRF00)_To_NAD_1983_2011'
        arcpy.Project_management(tempMergedFeatureClass,
outputGDBFeatureClass, outSR,transform); messages.addGPMessages()
#NEED TRANSFORMATION????

```

```

        arcpy.RepairGeometry_management(outputGDBFeatureClass);
messages.addGPMessages()

arcpy.MetadataImporter_conversion(os.path.join(parameters[4].valueAsText, metadataTemplate[0]), outputGDBFeatureClass);
messages.addGPMessages()

        arcpy.FeatureToPoint_management(outputGDBFeatureClass,
outputGDBFeatureClassPt,'INSIDE'); messages.addGPMessages()

arcpy.MetadataImporter_conversion(os.path.join(parameters[4].valueAsText, metadataTemplate[1]), outputGDBFeatureClassPt);
messages.addGPMessages()

        itemsToDelete = [tempLayer, mergeLayer, altLayer, affLayer,
tempDissolveFeatureClass, tempMergedFeatureClass,
outputLandsFeatureClass]
        for item in itemsToDelete:
            if arcpy.Exists(item):
                arcpy.Delete_management(item);
messages.addGPMessages()

```

Appendix C. Python Toolbox Code Processing I&M Operational Bounds

The following includes python toolbox code for processing and creating IMD_BND_Units.gdb.

```
import arcpy
import os, time
from datetime import datetime

class Toolbox(object):
    def __init__(self):
        """Define the toolbox (the name of the toolbox is the name
of the
        .pyt file)."""
        self.label = "IMDBounds_Processing"
        self.alias = ""

        # List of tool classes associated with this toolbox
        self.tools = [UpdateIMDBounds, IMDBoundsForServices]

class UpdateIMDBounds(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)."""
        self.label = "1 Update IMD Bounds"
        self.description = "Create or update IMD Bounds feature
classes in IMDBounds.gdb"
        self.canRunInBackground = False

    def getParameterInfo(self):
        """Define parameter definitions"""

        param0 = arcpy.Parameter(
            displayName = "Source Unit Bounds Feature Class
(IMD_BND_NPSUnits_wgs_py)",
            name = "unitBounds",
            datatype = "GPFeatureLayer",
            parameterType = "Required",
            direction = "Input")
        #param0.value =

        param1 = arcpy.Parameter(
            displayName = "Output GeoDatabase (IMD_Bounds.gdb)",
            name = "outDB",
```

```

        datatype = "DEWorkspace",
        parameterType = "Required",
        direction = "Input")

param2 = arcpy.Parameter(
    displayName = "IMD Bounds Lookup Table",
    name = "altBounds",
    datatype = "DETable",
    parameterType = "Required",
    direction = "Input")
#param2.value = ""

param3 = arcpy.Parameter(
    displayName = "Folder Containing Metadata Templates",
    name = "metaFolder",
    datatype = "Folder",
    parameterType = "Required",
    direction = "Input")
param3.value = ""

param4 = arcpy.Parameter(
    displayName = "Create Date (Current Date)",
    name = "CreateDate",
    datatype = "GPString",
    parameterType = "Optional",
    direction = "Input")
param4.value = "YYYYMMDD"

param5 = arcpy.Parameter(
    displayName = "Changed Units (List)",
    name = "Changed",
    datatype = "GPString",
    parameterType = "Optional",
    direction = "Input")
param5.value = "'PPPP','PPPP','PPPP'"

param6 = arcpy.Parameter(
    displayName = "Change Reason Domain Table
(EnvironmentalSettings.DBO_inpniscvdbnrst_IMDGIS.sde)",
    name = "ChangeDomain",
    datatype = "GPString",
    parameterType = "Required",
    direction = "Input")

```

```

#param6.value = ""

param7 = arcpy.Parameter(
    displayName = "Output Folder",
    name = "outLocation",
    datatype = "DEFolder",
    parameterType = "Required",
    direction = "Input")
#param7.value =

    params = [param0, param1, param2,
param3,param4,param5,param6,param7]
    return params

def isLicensed(self):
    """Set whether tool is licensed to execute."""
    return True

def updateParameters(self, parameters):
    """Modify the values and properties of parameters before
internal
validation is performed. This method is called whenever a
parameter
has been changed."""
    return

def updateMessages(self, parameters):
    """Modify the messages created by internal validation for
each tool
parameter. This method is called after internal
validation."""
    return

def projectDataset(self, sourceDataset, outputDataset,
outputSpatialRef, transform=None):
    """
    Re-projects and repairs source dataset
    """
    arcpy.Project_management(sourceDataset, outputDataset,
outputSpatialRef, transform)
    arcpy.RepairGeometry_management(outputDataset)
    arcpy.AddMessage('\n Re-projected to ' + outputDataset)

```

```

    def populateDatasets(self, targetLayer, targetDataset,
selectClause, outputFC, prevFC):
        arcpy.SelectLayerByAttribute_management(targetLayer,
"CLEAR_SELECTION")
        arcpy.SelectLayerByAttribute_management(targetLayer,
"NEW_SELECTION", "UNIT_CODE IN (" + selectClause + ")")
        arcpy.CopyFeatures_management(targetLayer, outputFC)
        arcpy.SelectLayerByAttribute_management(targetLayer,
"CLEAR_SELECTION")
        arcpy.RepairGeometry_management(outputFC)

arcpy.JoinField_management(outputFC, "UNIT_CODE", prevFC, "UNIT_CODE",
"CreateDate")
        arcpy.AddMessage('\n Created ' + outputFC)

    def updateNames(self, targetLayer, selectClause, targetFields):
        tempNameLayer = "tempNameLayer"
        arcpy.MakeFeatureLayer_management(targetLayer,
tempNameLayer)
        arcpy.SelectLayerByAttribute_management(tempNameLayer,
"NEW_SELECTION", "UNIT_CODE = '" + selectClause + "'")
        for targetField in targetFields:
            updatedValue = "!" + targetField + "! + ' and
Preserve'"
            arcpy.CalculateField_management(tempNameLayer,
targetField, updatedValue, "PYTHON_9.3")
            arcpy.SelectLayerByAttribute_management(tempNameLayer,
"CLEAR_SELECTION")
            arcpy.AddMessage('\n Updated Names: ' + selectClause + '
for ' + targetLayer)

    def execute(self, parameters, messages):
        message = ""
        today=datetime.now()
        datestamp = str(today.isoformat()).replace('-', '')[0:8]
        metadataTemplates = ["ES_IMDBoundsTemplate_FDGC.xml",
"ES_IMDBoundsTemplate_FDGC.xml",
"ES_IMDBoundsTemplate_FDGC.xml", "ES_IMDBoundsPtTemplate_FDGC.xml"]

        arcpy.env.overwriteOutput = 1
        # Lands bounds have M and Z values for some reason; this is
an extra check

```

```

        # since M and Z are disabled in the UnitBounds_Processing
UpdateUnitBounds tool
        arcpy.env.outputMFlag = "Disabled"
        arcpy.env.outputZFlag = "Disabled"
        outputInitTempFeatureClassName = "temp_unit_bounds_wgs0"
        outputTempFeatureClassName = "temp_unit_bounds_wgs"
        outputTempDissolveFeatureClassName =
"temp_dissolved_unit_bounds_wgs"
        outputGDBFeatureClassName = "IMD_BND_Units"
        outputGDBFeatureClassNameSvc = "IMD_BND_Units"
        tempLayer = "tempLayer"
        lookupTable = "lookupTable"
        createDate = parameters[4].valueAsText
        if createDate == 0:
            createDate = datestamp
        arcpy.AddMessage("DATE: " +createDate)
        selectUnits = parameters[5].valueAsText
        outputGDBFeatureClassInitTemp =
os.path.join(parameters[1].valueAsText,
outputInitTempFeatureClassName)
        outputGDBFeatureClassTemp =
os.path.join(parameters[1].valueAsText, outputTempFeatureClassName)
        outputGDBFeatureClassTempDissolve =
os.path.join(parameters[1].valueAsText,
outputTempDissolveFeatureClassName)
        outputGDBFeatureClassAlbers =
os.path.join(parameters[1].valueAsText, outputGDBFeatureClassName +
"_albers_py")
        outputGDBFeatureClassWebMerc =
os.path.join(parameters[1].valueAsText,
outputGDBFeatureClassNameSvc + "_webmercator_py")
        outputGDBFeatureClassWebMercPoint =
os.path.join(parameters[1].valueAsText,
outputGDBFeatureClassNameSvc + "_webmercator_pt")
        outputGDBFeatureClassConvexHulls =
os.path.join(parameters[1].valueAsText,
outputGDBFeatureClassNameSvc + "_webmercator_convexhulls")
        outputGDBFeatureClassWGS =
os.path.join(parameters[1].valueAsText,
outputGDBFeatureClassNameSvc + "_wgs_py")
        outputGDBFeatureClassWGSPoint =
os.path.join(parameters[1].valueAsText,
outputGDBFeatureClassNameSvc + "_wgs_pt")

```

```

        outputGDBFeatureClassAlaska =
os.path.join(parameters[1].valueAsText, "Alaska\\" +
outputGDBFeatureClassName + "_Alaska")
        outputGDBFeatureClassHawaii =
os.path.join(parameters[1].valueAsText, "Hawaii\\" +
outputGDBFeatureClassName + "_Hawaii")
        outputGDBFeatureClassSaipan =
os.path.join(parameters[1].valueAsText, "Pacific_Saipan_Guam\\" +
outputGDBFeatureClassName + "_Saipan_Guam")
        outputGDBFeatureClassSamoa =
os.path.join(parameters[1].valueAsText, "Pacific_Samoa\\" +
outputGDBFeatureClassName + "_Samoa")
        outputGDBFeatureClassCONUS =
os.path.join(parameters[1].valueAsText, "CONUS\\" +
outputGDBFeatureClassName + "_CONUS")

        outSRAlbers =
'PROJCS["USA_Contiguous_Albers_Equal_Area_Conic_USGS_version",GEOGC
S["GCS_North_American_1983",DATUM["D_North_American_1983",SPHEROID[
"GRS_1980",6378137.0,298.257222101]],PRIMEM["Greenwich",0.0],UNIT["
Degree",0.0174532925199433]],PROJECTION["Albers"],PARAMETER["False_
Easting",0.0],PARAMETER["False_Northing",0.0],PARAMETER["Central_Me
ridian",-
96.0],PARAMETER["Standard_Parallel_1",29.5],PARAMETER["Standard_Par
allel_2",45.5],PARAMETER["Latitude_Of_Origin",23.0],UNIT["Meter",1.
0]]", "NAD_1983_To_WGS_1984_4",
"GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID["WGS_1984",63781
37.0,298.257223563]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.017453
2925199433]]") '
        outSRWebMerc =
'PROJCS["WGS_1984_Web_Mercator_Auxiliary_Sphere",GEOGCS["GCS_WGS_19
84",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]
],PRIMEM["Greenwich",0.0],UNIT["Degree",0.017453292519943295]],PROJ
ECTION["Mercator_Auxiliary_Sphere"],PARAMETER["False_Easting",0.0],
PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridian",0.0],P
ARAMETER["Standard_Parallel_1",0.0],PARAMETER["Auxiliary_Sphere_Typ
e",0.0],UNIT["Meter",1.0]] '
        outSRWGS = arcpy.SpatialReference(4326)
        transformMethod = "WGS_1984_(ITRF00)_To_NAD_1983"

        # add CRMM
        AllUnits = [

```

'ABLI', 'ACAD', 'AGFO', 'ALAG', 'ALFL', 'ALKA', 'ALPO',
 'AMIS', 'AMME', 'ANIA', 'ANTI', 'APPA', 'APCO', 'APIS', 'ARCH',
 'ARPO', 'ASIS', 'AZRU',
 'BADL', 'BAND', 'BELA', 'BEOL', 'BIBE', 'BICA', 'BICY',
 'BIHO', 'BISC', 'BISO', 'BITH', 'BLCA', 'BLRI', 'BLUE', 'BOHA',
 'BOWA',
 'BRCA', 'BUFF', 'BUIS', 'CABR', 'CACH', 'CACO', 'CAGR',
 'CAHA', 'CAHE', 'CAKR', 'CALO', 'CAMO', 'CANA', 'CANY', 'CARE',
 'CARL', 'CASA', 'CATO',
 'CAVE', 'CAVO', 'CEBR', 'CHAT', 'CHCH', 'CHCU', 'CHIC',
 'CHIR', 'CHIS', 'CHOH', 'CHPI', 'CIRO', 'COLM', 'COLO', 'CORO',
 'CONG', 'COWP',
 'CRLA', 'CRMO', 'CUGA', 'CUIS', 'CURE', 'CUVA', 'DENA',
 'DEPO', 'DETO', 'DEVA', 'DEWA', 'DINO', 'DRTO', 'EBLA', 'EFMO',
 'EISE', 'ELMA',
 'ELMO', 'EUON', 'EVER', 'FIIS', 'FLFO', 'FOBO', 'FOBU',
 'FOCA', 'FODA', 'FODO', 'FOFR', 'FOLA', 'FOLS', 'FOMA', 'FOMO',
 'FONE',
 'FOPO', 'FOPU', 'FORA', 'FOSU', 'FOUN', 'FOUS', 'FOVA',
 'FOWA', 'FRDO', 'FRHI', 'FRSP', 'GAAR', 'GARI', 'GATE', 'GETT',
 'GEWA', 'GICL',
 'GLAC', 'GLBA', 'GLCA', 'GOGA', 'GOSP', 'GRBA', 'GRCA',
 'GREE', 'GRKO', 'GRPO', 'GRSA', 'GRSM', 'GRTE', 'GUCO', 'GUIS',
 'GUMO', 'GWCA',
 'GWMP', 'HAFE', 'HAFO', 'HALE', 'HAVO', 'HEHO', 'HOBE',
 'HOCU', 'HOFU', 'HOME', 'HONO', 'HOSP', 'HOVE', 'HUTR', 'INDU',
 'ISRO',
 'JECA', 'JELA', 'JODA', 'JOFL', 'JOMU', 'JOTR', 'KAHO',
 'KALA', 'KATM', 'KAWW', 'KEFJ', 'KEMO', 'KIMO', 'KLGO', 'KNRI',
 'KOVA', 'LABE',
 'LACL', 'LAKE', 'LAMR', 'LARO', 'LAVO', 'LEWI', 'LIBI',
 'LIBO', 'LIRI', 'LYJO', 'MABI', 'MACA', 'MANA', 'MANZ', 'MEVE',
 'MIMA',
 'MISS', 'MNRR', 'MOCA', 'MOCR', 'MOJA', 'MONO', 'MORA',
 'MORR', 'MORU', 'MUWO', 'NABR', 'NACE', 'NATR', 'NAVA', 'NEPE',
 'NERI', 'NIOB',
 'NISI', 'NOAT', 'NOCA', 'NPSA', 'OBRI', 'OCMU', 'OLYM',
 'ORCA', 'ORPI', 'OZAR', 'PAAL', 'PAIS', 'PARA', 'PECO', 'PEFO',
 'PERI', 'PETE', 'PETR',
 'PINN', 'PIPE', 'PIRO', 'PISP', 'POCH', 'PORE', 'PRWI',
 'PUHE', 'PUHO', 'RABR', 'REDW', 'RICH', 'RIGR', 'ROCR', 'ROMO',
 'ROVA', 'RUCA', 'SAAN', 'SACN', 'SAGA', 'SAGU', 'SAHI',
 'SAIR', 'SAJH', 'SAMO', 'SAND', 'SAPU', 'SARA', 'SCBL', 'SEKI',

```

        'SHEN', 'SHIL', 'SITK', 'SLBE', 'STRI', 'SUCR', 'TAPR',
        'THRO', 'THST', 'TICA', 'TIMU', 'TONT', 'TUMA', 'TUSK', 'TUZI',
        'UPDE', 'VALR', 'VAFO',
        'VICK', 'VIIS', 'VOYA', 'WABA', 'WACA', 'WAPA', 'WEFA',
        'WHIS', 'WHMI', 'WHS', 'WICA', 'WICR', 'WOTR', 'WRBR', 'WRST',
        'WUPA',
        'YELL', 'YOSE', 'YUCH', 'YUHO', 'ZION']

```

```

    Alaska_Parks =
    "'ALAG','ANIA','BELA','CAKR','DENA','GAAR','GLBA','KATM','KEFJ','KO
    VA','KLGO','LACL','NOAT','SITK','YUCH','WRST'"
    Hawaii_Parks =
    "'ALKA','HALE','KALA','KAHO','HAVO','HONO','PUHE','PUHO','VALR'"
    PacificSaipan_Parks = "'AMME','WAPA'"
    PacificSamoa_Parks= "'NPSA'"
    Alaska_Parks_Array =
    ['ALAG','ANIA','BELA','CAKR','DENA','GAAR','GLBA','KATM','KEFJ','KO
    VA','KLGO','LACL','NOAT','SITK','YUCH','WRST']
    Hawaii_Parks_Array =
    ['ALKA','HALE','KALA','KAHO','HAVO','HONO','PUHE','PUHO','VALR']
    PacificSaipan_Parks_Array = ['AMME','WAPA']
    PacificSamoa_Parks_Array = ['NPSA']
    Conus_Select_Array0 =
    [Alaska_Parks_Array,Hawaii_Parks_Array,PacificSaipan_Parks_Array,Pa
    cificSamoa_Parks_Array]
    Conus_Select_Array = [ y for x in Conus_Select_Array0 for y
    in x]
    for unit in Conus_Select_Array:
        AllUnits.remove(unit)

    UpdateNames_Array =
    ['ALAG','ANIA','DENA','GAAR','GLBA','KATM','LACL','WRST','GRSA','CR
    MO','CRMM']

    # Join lookup to unit_bounds_wgs and copy matching features
    to temp feature class

    arcpy.FeatureClassToFeatureClass_conversion(parameters[0].valueAsText,
    parameters[1].valueAsText, outputInitTempFeatureClassName);
    messages.addGPMessages()

```

```

arcpy.MakeFeatureLayer_management(outputGDBFeatureClassInitTemp,
tempLayer); messages.addGPMessages()
    # Change CRMG back to CRMO for Dissolve

arcpy.SelectLayerByAttribute_management(tempLayer,"NEW_SELECTION",
UNIT_CODE = 'CRMG'); messages.addGPMessages()

arcpy.CalculateField_management(tempLayer,"UNIT_CODE",'CRMO');
messages.addGPMessages()

arcpy.SelectLayerByAttribute_management(tempLayer,"CLEAR_SELECTION"
); messages.addGPMessages()
    #
    arcpy.MakeTableView_management(parameters[2].valueAsText,
lookupTable); messages.addGPMessages()
    arcpy.JoinField_management(tempLayer, "UNIT_CODE",
lookupTable, "Code", ["NetworkCode", "Lifecycle"]);
messages.addGPMessages()
    arcpy.SelectLayerByAttribute_management(tempLayer,
"NEW_SELECTION", "NetworkCode IS NOT NULL");
messages.addGPMessages()
    arcpy.Dissolve_management(tempLayer,
outputGDBFeatureClassTempDissolve, "UNIT_CODE");
messages.addGPMessages()

arcpy.RepairGeometry_management(outputGDBFeatureClassTempDissolve);
messages.addGPMessages()

arcpy.AddField_management(outputGDBFeatureClassTempDissolve,
"Network_Code", "TEXT", "", "", 4,"","","","");
messages.addGPMessages()

arcpy.JoinField_management(outputGDBFeatureClassTempDissolve,
"UNIT_CODE", tempLayer, "UNIT_CODE",
["UNIT_NAME","DATE_EDIT","Unit_Code_Display","Full_Name","Feature_S
ource","Source_Metadata","Identifier","NetworkCode","CreateDate","R
edactReason","ChangeReason","DSLLink","UnitCodeQualifier","Processin
gQualifier","DataCertificationLevel","Lifecycle"]);
messages.addGPMessages()

```

```

arcpy.CopyFeatures_management(outputGDBFeatureClassTempDissolve,
outputGDBFeatureClassTemp); messages.addGPMessages()
    arcpy.CalculateField_management(outputGDBFeatureClassTemp,
"Network_Code", "!NetworkCode!", "PYTHON"); messages.addGPMessages()

arcpy.DeleteField_management(outputGDBFeatureClassTemp, "NetworkCode
"); messages.addGPMessages()
    arcpy.CalculateField_management(outputGDBFeatureClassTemp,
"Feature_Source", "NPS Inventory and Monitoring Division - Central
Office", "PYTHON"); messages.addGPMessages()
    arcpy.CalculateField_management(outputGDBFeatureClassTemp,
"Source_Metadata", "!DSLink!", "PYTHON"); messages.addGPMessages()

arcpy.CalculateField_management(outputGDBFeatureClassTemp, "DSLink",
'"https://irma.nps.gov/DataStore/Reference/Profile/2245954?lnv=True
"'); messages.addGPMessages()

arcpy.CalculateField_management(outputGDBFeatureClassTemp, "FeatureQ
ualifier", '"NC"', "PYTHON"); messages.addGPMessages()

arcpy.CalculateField_management(outputGDBFeatureClassTemp, "UnitCode
Qualifier", '"SC"', "PYTHON"); messages.addGPMessages()

    # Update names for logical (dissolved) parks
    for unitCode in UpdateNames_Array:
        UpdateIMDBounds.updateNames(self,
outputGDBFeatureClassTemp, unitCode, ["UNIT_NAME", "Full_Name"])

    # Delete older versions
    fcsToDelete = [outputGDBFeatureClassName + '_albers_py' +
'_2*',
                    outputGDBFeatureClassNameSvc + '_wgs_py' +
'_2*',
                    outputGDBFeatureClassWebMercPoint,
                    outputGDBFeatureClassWGSPoint]

    workspace = parameters[1].valueAsText
    arcpy.env.workspace = workspace
    for fc in fcsToDelete:
        fcs = arcpy.ListFeatureClasses(fc, "Polygon");
messages.addGPMessages()

```

```

        for fcc in fcs:
            arcpy.Delete_management(fcc);
messages.addGPMessages()

# Version existing feature classes; Project to new feature
classes
fcsToVersion = [outputGDBFeatureClassAlbers,
                 outputGDBFeatureClassConvexHulls,
                 outputGDBFeatureClassNameSvc,
                 outputGDBFeatureClassWGS]

for fc in fcsToVersion:
    if arcpy.Exists(fc):
        arcpy.Rename_management(fc, fc + '_' + datestamp);
messages.addGPMessages()

arcpy.CalculateField_management(outputGDBFeatureClassTemp, "Unit_Cod
e_Display", "!UNIT_CODE!", "PYTHON"); messages.addGPMessages()

arcpy.MakeFeatureLayer_management(outputGDBFeatureClassTemp, "fcTemp
"); messages.addGPMessages()

arcpy.SelectLayerByAttribute_management("fcTemp", "NEW_SELECTION", "U
NIT_CODE = 'ANIA' OR UNIT_CODE = 'CRMO' OR UNIT_CODE = 'DENA' OR
UNIT_CODE = 'GAAR' OR UNIT_CODE = 'GLBA' OR UNIT_CODE = 'GRSA' OR
UNIT_CODE = 'KATM' OR UNIT_CODE = 'LACL' OR UNIT_CODE = 'WRST'");
messages.addGPMessages()

arcpy.CalculateField_management("fcTemp", "FeatureQualifier", '"MS"',
"PYTHON"); messages.addGPMessages()

arcpy.CalculateField_management("fcTemp", "Identifier", '"IMD-Park-
4326-!Unit_Code_Display!""!CreateDate!',"PYTHON");
messages.addGPMessages()

arcpy.SelectLayerByAttribute_management("fcTemp", "CLEAR_SELECTION")
; messages.addGPMessages()

UpdateIMDBounds.projectDataset(self,
outputGDBFeatureClassTemp, outputGDBFeatureClassAlbers,
outSRAlbers, transformMethod)

```

```

        UpdateIMDBounds.projectDataset(self,
outputGDBFeatureClassTemp, outputGDBFeatureClassWebMerc,
outSRWebMerc)

arcpy.CalculateField_management(outputGDBFeatureClassAlbers,"Identifier",'"IMD-Park-102039-"!Unit_Code_Display!'"-
"!CreateDate!',"PYTHON"); messages.addGPMessages()

arcpy.CopyFeatures_management(outputGDBFeatureClassTemp,outputGDBFeatureClassWGS); messages.addGPMessages()

        # Create convex hull feature class

arcpy.MinimumBoundingGeometry_management(outputGDBFeatureClassWebMerc, outputGDBFeatureClassConvexHulls, "CONVEX_HULL");
messages.addGPMessages()

arcpy.RepairGeometry_management(outputGDBFeatureClassConvexHulls);
messages.addGPMessages()

        # Create feature classes in feature datasets:

arcpy.MakeFeatureLayer_management(outputGDBFeatureClassTemp,
tempLayer); messages.addGPMessages()

        UpdateIMDBounds.populateDatasets(self, tempLayer, "Alaska",
Alaska_Parks, outputGDBFeatureClassAlaska,outputGDBFeatureClassName
+ '_albers_py_' + datestamp)
        UpdateIMDBounds.populateDatasets(self, tempLayer, "Hawaii",
Hawaii_Parks, outputGDBFeatureClassHawaii,outputGDBFeatureClassName
+ '_albers_py_' + datestamp)
        UpdateIMDBounds.populateDatasets(self, tempLayer,
"Pacific_Saipan_Guam", PacificSaipan_Parks,
outputGDBFeatureClassSaipan,outputGDBFeatureClassName +
'_albers_py_' + datestamp)
        UpdateIMDBounds.populateDatasets(self, tempLayer,
"Pacific_Samoa", PacificSamoa_Parks,
outputGDBFeatureClassSamoa,outputGDBFeatureClassName +
'_albers_py_' + datestamp)

```

```

        arcpy.SelectLayerByAttribute_management(tempLayer,
"NEW_SELECTION", "UNIT_CODE IN (" + Alaska_Parks + ")");
messages.addGPMessages()
        arcpy.SelectLayerByAttribute_management(tempLayer,
"ADD_TO_SELECTION", "UNIT_CODE IN (" + Hawaii_Parks + ")");
messages.addGPMessages()
        arcpy.SelectLayerByAttribute_management(tempLayer,
"ADD_TO_SELECTION", "UNIT_CODE IN (" + PacificSaipan_Parks + ")");
messages.addGPMessages()
        arcpy.SelectLayerByAttribute_management(tempLayer,
"ADD_TO_SELECTION", "UNIT_CODE IN (" + PacificSamoa_Parks + ")");
messages.addGPMessages()
        arcpy.SelectLayerByAttribute_management(tempLayer,
"SWITCH_SELECTION"); messages.addGPMessages()
        arcpy.CopyFeatures_management(tempLayer,
outputGDBFeatureClassCONUS); messages.addGPMessages()

        arcpy.SelectLayerByAttribute_management(tempLayer,
"CLEAR_SELECTION"); messages.addGPMessages()

arcpy.RepairGeometry_management(outputGDBFeatureClassCONUS);
messages.addGPMessages()

arcpy.JoinField_management(outputGDBFeatureClassCONUS, "UNIT_CODE", o
utputGDBFeatureClassName + '_albers_py_' +
datestamp, "UNIT_CODE", "CreateDate"); messages.addGPMessages()

        # Import metadata, edit field CreateDate, add Domain to
feature classes
        for fc in fcsToVersion:
            if arcpy.Exists(fc):

arcpy.AssignDomainToField_management(fc, "ChangeReason", "ChangeReaso
n_Domain"); messages.addGPMessages()
                arcpy.MakeFeatureLayer_management(fc, tempLayer);
messages.addGPMessages()
                arcpy.SelectLayerByAttribute_management(tempLayer,
"NEW_SELECTION", "UNIT_CODE IN (" + selectUnits + ")");
messages.addGPMessages()
                result = arcpy.GetCount_management(tempLayer);
messages.addGPMessages()
                count = int(result.getOutput(0))
                if count != 0:

```

```

arcpy.CalculateField_management(tempLayer,"CreateDate",''+createDate+''); messages.addGPMessages()
        arcpy.SelectLayerByAttribute_management(tempLayer,
"CLEAR_SELECTION", "UNIT_CODE IN ('"+selectUnits+"")");
messages.addGPMessages()
        if fc.endswith("_albers_py"):

arcpy.MetadataImporter_conversion(os.path.join(parameters[3].valueAsText, metadataTemplates[0]), fc); messages.addGPMessages()
        elif fc.endswith("_wgs_py"):

arcpy.MetadataImporter_conversion(os.path.join(parameters[3].valueAsText, metadataTemplates[1]), fc); messages.addGPMessages()
        else:
            pass

        flds =
["Identifier", "UnitCodeQualifier", "FeatureQualifier", "RedactReason",
"ChangeReason"]
        for fld in flds:

arcpy.DeleteField_management(outputGDBFeatureClassConvexHulls, fld);
messages.addGPMessages()
            if arcpy.Exists:

arcpy.Delete_management(os.path.join(outputGDBFeatureClassConvexHulls, datestamp))

        arcpy.FeatureToPoint_management(outputGDBFeatureClassWGS,
outputGDBFeatureClassWGSPoint, "INSIDE"); messages.addGPMessages()

arcpy.CalculateField_management(outputGDBFeatureClassWGSPoint, "Identifier", "'IMD-ParkPt-4326-'!Unit_Code_Display!'"-
"!CreateDate!'", "PYTHON"); messages.addGPMessages()

arcpy.MetadataImporter_conversion(os.path.join(parameters[3].valueAsText, metadataTemplates[3]), outputGDBFeatureClassWGSPoint);
messages.addGPMessages()

        fcsInDatasets = [outputGDBFeatureClassAlaska,
outputGDBFeatureClassHawaii, outputGDBFeatureClassSaipan,
outputGDBFeatureClassSamoa, outputGDBFeatureClassCONUS]

```

```

        for fc in fcsInDatasets:

arcpy.AssignDomainToField_management(fc,"ChangeReason","ChangeReason_Domain"); messages.addGPMessages()

arcpy.MetadataImporter_conversion(os.path.join(parameters[3].valueAsText, metadataTemplates[2]), fc); messages.addGPMessages()
        arcpy.MakeFeatureLayer_management(fc,tempLayer);
messages.addGPMessages()
        arcpy.SelectLayerByAttribute_management(tempLayer,
"NEW_SELECTION", "UNIT_CODE IN ('+selectUnits+')");
messages.addGPMessages()
        result = arcpy.GetCount_management(tempLayer);
messages.addGPMessages()
        count = int(result.getOutput(0))
        if count != 0:

arcpy.CalculateField_management(tempLayer,"CreateDate",''+createDate+''); messages.addGPMessages()
        arcpy.SelectLayerByAttribute_management(tempLayer,
"CLEAR_SELECTION"); messages.addGPMessages()

        itemsToDelete = [tempLayer, lookupTable,
outputGDBFeatureClassTemp, outputGDBFeatureClassInitTemp,
outputGDBFeatureClassTempDissolve, outputGDBFeatureClassWebMerc]
        for item in itemsToDelete:
            if arcpy.Exists(item):
                arcpy.Delete_management(item);
messages.addGPMessages()

        # Create output dataset for publication: IMD_BND_Units.gdb
arcpy.AddMessage("\n Create geodatabase dataset for
publication \n")
        pubNameGDB = "IMD_BND_Units.gdb"
        pubFolder = parameters[7].valueAsText
        pubPath = (os.path.join(pubFolder,pubNameGDB))
        Albers = os.path.join(pubPath,"IMD_BND_Units_albers_py")
        WGS = os.path.join(pubPath,"IMD_BND_Units_wgs_py")
        WGSpt = os.path.join(pubPath,"IMD_BND_Units_wgs_pt")

        if arcpy.Exists(pubPath):
            pass
        else:

```

```

        arcpy.CreateFileGDB_management(pubFolder, pubNameGDB);
messages.addGPMessages()

arcpy.CopyFeatures_management(outputGDBFeatureClassWGS, WGS);
messages.addGPMessages()

arcpy.CopyFeatures_management(outputGDBFeatureClassWGSPoint, WGSpt);
messages.addGPMessages()

arcpy.CopyFeatures_management(outputGDBFeatureClassAlbers, Albers);
messages.addGPMessages()

```

```

class IMDBoundsForServices(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)."""
        self.label = "2 IMD Bounds for Services"
        self.description = "Create or update IMD Bounds feature
classes in GDB_IMD.DBO.INPNISCVDBNRSST_IMDGIS.sde"
        self.canRunInBackground = False

    def getParameterInfo(self):
        """Define parameter definitions"""

        param0 = arcpy.Parameter(
            displayName = "WGS IMD Unit Bounds Feature Class",
            name = "unitBounds",
            datatype = "GPFeatureLayer",
            parameterType = "Required",
            direction = "Input")
        param0.value = r""

        param1 = arcpy.Parameter(
            displayName = "Replace Feature",
            name = "APPA",
            datatype = "GPFeatureLayer",
            parameterType = "Required",
            direction = "Input")
        param1.value = r""

        param2 = arcpy.Parameter(

```

```

        displayName = "Convex Hulls Unit Bounds Feature Class",
        name = "unitBoundsCH",
        datatype = "GPFeatureLayer",
        parameterType = "Required",
        direction = "Input")
    param2.value = r""

    param3 = arcpy.Parameter(
        displayName = "GDB_IMD Database Connection
(GDB_IMD.DBO.INPNISCVDBNRSST_IMDGIS.sde) ",
        name = "DB",
        datatype = "DEWorkspace",
        parameterType = "Optional",
        direction = "Input")
    param4.value = "Connection here"

    param4 = arcpy.Parameter(
        displayName = "NPSData Database Connection
(NPSData.DBO.INPNISCVDBNRSST_IMDGIS.sde) ",
        name = "DB2",
        datatype = "DEWorkspace",
        parameterType = "Optional",
        direction = "Input")
    param5.value = "Connection here"

    param5 = arcpy.Parameter(
        displayName = "Output Folder",
        name = "outLocation",
        datatype = "DEFolder",
        parameterType = "Required",
        direction = "Input")
    #param5.value =

    param6 = arcpy.Parameter(
        displayName = "Update SDE",
        name = "sde",
        datatype = "Boolean",
        parameterType = "Optional",
        direction = "Input")

    params = [param0, param1, param2, param3, param4, param5,
param6]
    return params

```

```

def isLicensed(self):
    """Set whether tool is licensed to execute."""
    return True

def updateParameters(self, parameters):
    """Modify the values and properties of parameters before
internal
validation is performed. This method is called whenever a
parameter
has been changed."""
    return

def updateMessages(self, parameters):
    """Modify the messages created by internal validation for
each tool
parameter. This method is called after internal
validation."""
    return

def execute(self, parameters, messages):
    message = ""
    today=datetime.now()
    datestamp = str(today.isoformat()).replace('-', '')[0:8]

    arcpy.env.overwriteOutput = 1
    SDE = (parameters[3].valueAsText)
    SDE1 = (parameters[4].valueAsText)
    GDBFeatureClassName = "IMD_BND_UnitsService"
    GDBFeatureClassWGS = (parameters[0].valueAsText)
    GDBFeatureClassConvexHulls = (parameters[2].valueAsText)
    GDBFeatures =
r"X:\ProjectData\Data_Processing\Bounds_Processing\Data\IMD_Bounds.
gdb"
    outputSDEFeatureClassConvexHulls = (SDE + "\\\" +
GDBFeatureClassName + "_webmercator_convexhulls")
    outputSDEFeatureClassWGS =
os.path.join(SDE,GDBFeatureClassName+'_wgs_py')
    outputFolder = parameters[5].valueAsText
    pubNameGDB = "IMD_BND_Units.gdb"
    pubPath = (os.path.join(outputFolder,pubNameGDB))
    updateSDE = parameters[6].valueAsText

```

```

        outputSDEFeatureClassNPSData = (SDE1 + "\\\" +
GDBFeatureClassName + "Service_wgs_py")

        APPA_generalized = (parameters[1].valueAsText)
        temp =
r"X:\ProjectData\Data_Processing\Bounds_Processing\Data\Scratch.gdb
\Temp"
        tempLayer =
r"X:\ProjectData\Data_Processing\Bounds_Processing\Data\Scratch.gdb
\wgsTemp"
        tempFeature =
r"X:\ProjectData\Data_Processing\Bounds_Processing\Data\Scratch.gdb
\TempFeature"

        arcpy.CopyFeatures_management(GDBFeatureClassWGS, temp);
messages.addGPMessages()
        arcpy.MakeFeatureLayer_management(temp, tempLayer);
messages.addGPMessages()

        arcpy.SelectLayerByAttribute_management(tempLayer,
"NEW_SELECTION", "UNIT_CODE = 'APPA'"); messages.addGPMessages()
        if int(arcpy.GetCount_management(tempLayer).getOutput(0)) >
0:
            arcpy.DeleteFeatures_management(tempLayer);
messages.addGPMessages()
            arcpy.SelectLayerByAttribute_management(tempLayer,
"CLEAR_SELECTION"); messages.addGPMessages()

arcpy.Merge_management([tempLayer,APPA_generalized],tempFeature);
messages.addGPMessages()
        arcpy.RepairGeometry_management(tempFeature);
messages.addGPMessages()
        arcpy.MakeFeatureLayer_management(tempFeature,"ttemp");
messages.addGPMessages()

arcpy.SelectLayerByAttribute_management("ttemp","NEW_SELECTION","UN
IT_CODE = 'APPA'"); messages.addGPMessages()

arcpy.CalculateField_management("ttemp","Unit_Code_Display","!UNIT_
CODE!","PYTHON"); messages.addGPMessages()

```

```

        arcpy.CalculateField_management("ttemp", "Identifier", '"IMD-
ParkService-4326-"!Unit_code_Display!"-"!CreateDate!"', "PYTHON");
messages.addGPMessages()

arcpy.CalculateField_management("ttemp", "UnitCodeQualifier", '"SC"',
"PYTHON"); messages.addGPMessages()

arcpy.CalculateField_management("ttemp", "FeatureQualifier", '"RS"', "
PYTHON"); messages.addGPMessages()

arcpy.CalculateField_management("ttemp", "DataCertificationLevel", '"
C"', "PYTHON"); messages.addGPMessages()

arcpy.CalculateField_management("ttemp", "Unit_Code_Display", '"!UNIT_
CODE!"', "PYTHON"); messages.addGPMessages()

arcpy.CalculateField_management("ttemp", "Source_Metadata", '"https:/
/irma.nps.gov/DataStore/Reference/Profile/2245886?lnv=True"', "PYTHO
N"); messages.addGPMessages()

arcpy.SelectLayerByAttribute_management("ttemp", "CLEAR_SELECTION");
messages.addGPMessages()
        arcpy.DeleteField_management(tempFeature, "Source_Details");
messages.addGPMessages()

        if updateSDE == 'true':
            arcpy.AddMessage("Updating SDE Feature Classes...")
            features = [outputSDEFeatureClassConvexHulls]
            #Delete Features used here to prevent having to
recreate permissions for all data layers
            for feat in features:
                if arcpy.Exists(feat):
                    arcpy.DeleteFeatures_management(feat);
messages.addGPMessages()
                else:
                    pass

        FCs =
["IMD_BND_Units_webmercator_convexhulls", "IMD_BND_UnitsService_wgs_
py", "IMD_BND_UnitsService_wgs_pt"]

        for fc in FCs:

```

```

        arcpy.CopyFeatures_management(GDBFeatures+"\\ "+fc,
SDE+"\\ "+fc); messages.addGPMessages()

        arcpy.CopyFeatures_management(tempFeature,
outputSDEFeatureClassWGS); messages.addGPMessages()
        arcpy.CopyFeatures_management(tempFeature,
outputSDEFeatureClassNPSData); messages.addGPMessages()
    else:
        pass

    arcpy.CopyFeatures_management(tempFeature,
os.path.join(pubPath,"IMD_BND_UnitsService_wgs_py"));
messages.addGPMessages()

    itemsToDelete = [temp, tempLayer, tempFeature]
    for item in itemsToDelete:
        if arcpy.Exists(item): arcpy.Delete_management(item);
messages.addGPMessages()

```

Appendix D. Additional Processing Script for Adjusted NPS Unit Boundaries

Additional processing script to create posted output for Adjusted NPS Unit Boundaries.

```
import arcpy
import os, time
gdbToBeUpdated = r'' # this value is the file geodatabase of the
Adjusted NPS Unit Bounds polygon and point feature classes
pyToBeUpdated = r'' # this value is the polygon feature class from
Adjusted NPS Unit Bounds
ptToBeUpdated = r'' # this value is the point feature class from
Adjusted NPS Unit Bounds
DomainGDB = r'' # Domain GDB: see cited reference (NPS IMD 2018f)
DOM_Change = 'DOM_ChangeReasonFromPrecedingVersion_NPSIMD2019'
DOM_DataCert = 'DOM_DataCertificationLevel_NPSIMD2019'
DOM_EDITED = 'DOM_EDITEDBY_NPSIMD2019'
DOM_FeatAlt = 'DOM_FeatureAlteration_NPSIMD2019'
DOM_FeatLife = 'DOM_FeatureLifecycle_NPSIMD2019'
DOM_ORIG = 'DOM_ORIGINATOR_NPSIMD2019'
DOM_UnitCodeAlt = 'DOM_UnitCodeAlteration_NPSIMD2019'
arcpy.Delete_management(ptToBeUpdated)
arcpy.RemoveDomainFromField_management(pyToBeUpdated, "DataCertifica
tionLevel")
arcpy.RemoveDomainFromField_management(pyToBeUpdated, "FeatureQualif
ier")
arcpy.RemoveDomainFromField_management(pyToBeUpdated, "Feature_Sourc
e")
arcpy.RemoveDomainFromField_management(pyToBeUpdated, "Lifecycle")
arcpy.RemoveDomainFromField_management(pyToBeUpdated, "UnitCodeQuali
fier")
arcpy.RemoveDomainFromField_management(pyToBeUpdated, "ChangeReason"
)
arcpy.RemoveDomainFromField_management(pyToBeUpdated, "RedactReason"
)
arcpy.DeleteDomain_management(gdbToBeUpdated, "ChangeReason_Domain")
arcpy.DeleteDomain_management(gdbToBeUpdated, "DataCertificationLeve
l_Domain")
arcpy.DeleteDomain_management(gdbToBeUpdated, "FeatureQualifier_Doma
in")
arcpy.DeleteDomain_management(gdbToBeUpdated, "FeatureSource_Domain"
)
arcpy.DeleteDomain_management(gdbToBeUpdated, "Lifecycle_Domain")
```

```

arcpy.DeleteDomain_management(gdbToBeUpdated,"RedactReason_Domain")
arcpy.DeleteDomain_management(gdbToBeUpdated,"UnitCodeQualifier_Domain")
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"ChangeReason_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"DataCertificationLevel_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"FeatureQualifier_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"FeatureSource_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"Lifecycle_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"RedactReason_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"UnitCodeQualifier_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"Lifecycle"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"Feature_Source"))
arcpy.AlterField_management(pyToBeUpdated,"UNIT_CODE","UNITCODE",'UNITCODE',"TEXT",16,"NULLABLE",'None')
arcpy.AlterField_management(pyToBeUpdated,"UNIT_NAME","UNITNAME","UNITNAME","TEXT",254,"NULLABLE",'None')
arcpy.AlterField_management(pyToBeUpdated,"DATE_EDIT","EDITDATE","EDITDATE","DATE","", "NULLABLE", "None")
arcpy.AlterField_management(pyToBeUpdated,"Unit_Code_Display","UnitCodeDisplay","UnitCodeDisplay","TEXT",25,"NULLABLE",'None')
arcpy.AlterField_management(pyToBeUpdated,"Feature_Source","ORIGINATOR","ORIGINATOR","TEXT",100,"NULLABLE",'None')
arcpy.AlterField_management(pyToBeUpdated,"Source_Metadata","OriginatorMetadata","OriginatorMetadata","TEXT",150,"NULLABLE",'None')
arcpy.AlterField_management(pyToBeUpdated,"Source_Details","MAPSOURCE","MAPSOURCE","TEXT",500,"NULLABLE",'None')
arcpy.AlterField_management(pyToBeUpdated,"Identifier","GEOMETRYID","GEOMETRYID","TEXT",250,"NULLABLE",'None')
arcpy.AlterField_management(pyToBeUpdated,"CreateDate","IMDCreateDate","IMDCreateDate","TEXT",10,"NULLABLE",'None')
arcpy.AlterField_management(pyToBeUpdated,"DSLink","FeatureMetadata","FeatureMetadata","TEXT",150,"NULLABLE",'None')
arcpy.AlterField_management(pyToBeUpdated,"Lifecycle","FeatureLifecycle","FeatureLifecycle","TEXT",15,"NULLABLE",'None')

```

```

arcpy.AddField_management(pyToBeUpdated,"ChangeReasonFromPrecedingVersion","TEXT",'',50,"ChangeReasonFromPrecedingVersion","NULLABLE","NON_REQUIRED")
arcpy.CalculateField_management(pyToBeUpdated,"ChangeReasonFromPrecedingVersion",'!ChangeReason!',"PYTHON_9.3")
arcpy.DeleteField_management(pyToBeUpdated,"ChangeReason")
arcpy.DeleteField_management(pyToBeUpdated,"Full_Name")
arcpy.DeleteField_management(pyToBeUpdated,"RedactReason")
arcpy.AddField_management(pyToBeUpdated,"EDITEDBY","TEXT",'',100,"EDITEDBY","NULLABLE","NON_REQUIRED")
arcpy.AlterField_management(pyToBeUpdated,"UnitCodeQualifier","UnitCodeAlteration","UnitCodeAlteration","TEXT",10,'NULLABLE','None')
arcpy.AlterField_management(pyToBeUpdated,"FeatureQualifier","FeatureAlteration","FeatureAlteration","TEXT",10,"NULLABLE","NON_REQUIRED")
arcpy.JoinField_management(pyToBeUpdated,"UnitCodeAlteration",os.path.join(DomainGDB,DOM_UnitCodeAlt),"Code","UnitCodeAlteration_Label")
arcpy.JoinField_management(pyToBeUpdated,"FeatureAlteration",os.path.join(DomainGDB,DOM_FeatAlt),"Code","FeatureAlteration_Label")
arcpy.MakeFeatureLayer_management(pyToBeUpdated,"py_temp")
arcpy.SelectLayerByAttribute_management("py_temp","NEW_SELECTION","ChangeReasonFromPrecedingVersion = 'Spatial change (Lands)'"")
arcpy.CalculateField_management("py_temp","ChangeReasonFromPrecedingVersion","LRD","PYTHON_9.3")
arcpy.SelectLayerByAttribute_management("py_temp","SWITCH_SELECTION")
arcpy.CalculateField_management("py_temp","ChangeReasonFromPrecedingVersion","NC","PYTHON_9.3")
arcpy.SelectLayerByAttribute_management("py_temp","NEW_SELECTION","IMDCreateDate = '20181003'"")
arcpy.CalculateField_management("py_temp","ChangeReasonFromPrecedingVersion","LRD","PYTHON_9.3")
arcpy.SelectLayerByAttribute_management("py_temp","CLEAR_SELECTION")
arcpy.JoinField_management(pyToBeUpdated,"ChangeReasonFromPrecedingVersion",os.path.join(DomainGDB,DOM_Change),'Code',"ChangeReasonFromPrecedingVersion_Label")
arcpy.SelectLayerByAttribute_management("py_temp","CLEAR_SELECTION")

select = "UNITCODE = 'AMIS' OR UNITCODE = 'ASIS' OR UNITCODE = 'BEFR' OR UNITCODE = 'BICY' OR UNITCODE = 'BISO' OR UNITCODE = 'BLRI' OR UNITCODE = 'BLUE' OR UNITCODE = 'CAHA' OR UNITCODE =

```

```
'CALO' OR UNITCODE = 'CHAT' OR UNITCODE = 'CHCH' OR UNITCODE =
'CHRO' OR UNITCODE = 'CUIS' OR UNITCODE = 'EBLA' OR UNITCODE =
'EVER' OR UNITCODE = 'FIIS' OR UNITCODE = 'FORA' OR UNITCODE =
'GARI' OR UNITCODE = 'GRSP' OR UNITCODE = 'HATU' OR UNITCODE =
'ILMI' OR UNITCODE = 'ISRO' OR UNITCODE = 'LAMR' OR UNITCODE =
'LARO' OR UNITCODE = 'LEWI' OR UNITCODE = 'LOWE' OR UNITCODE =
'NEPE' OR UNITCODE = 'NERI' OR UNITCODE = 'OBRI' OR UNITCODE =
'POPO' OR UNITCODE = 'PRSF' OR UNITCODE = 'SAJH' OR UNITCODE =
'SAJU' OR UNITCODE = 'SAMO' OR UNITCODE = 'SHEN' OR UNITCODE =
'SHIL' OR UNITCODE = 'SLBE' OR UNITCODE = 'STEA' OR UNITCODE =
'TIMU' OR UNITCODE = 'VICK'"
```

```
arcpy.SelectLayerByAttribute_management("py_temp","NEW_SELECTION",s
elect)
arcpy.CalculateField_management("py_temp","ORIGINATOR","IMD-
Central Office","PYTHON_9.3")
arcpy.CalculateField_management("py_temp","OriginatorMetadata","ht
tps://irma.nps.gov/DataStore/Reference/Profile/2237295","PYTHON_9.
3")
arcpy.SelectLayerByAttribute_management("py_temp","CLEAR_SELECTION"
)
arcpy.CalculateField_management(pyToBeUpdated,"EDITEDBY","!ORIGINAT
OR!","PYTHON_9.3")
arcpy.DeleteField_management(pyToBeUpdated,"ChangeReason_Label")
arcpy.DeleteField_management(pyToBeUpdated,"FeatureQualifier_Label"
)
arcpy.DeleteField_management(pyToBeUpdated,"UnitCodeQualifier_Label
")
arcpy.CopyRows_management(os.path.join(DomainGDB,DOM_Change),os.pat
h.join(gdbToBeUpdated,DOM_Change))
arcpy.CopyRows_management(os.path.join(DomainGDB,DOM_DataCert),os.p
ath.join(gdbToBeUpdated,DOM_DataCert))
arcpy.CopyRows_management(os.path.join(DomainGDB,DOM_EDITED),os.pat
h.join(gdbToBeUpdated,DOM_EDITED))
arcpy.CopyRows_management(os.path.join(DomainGDB,DOM_FeatAlt),os.pa
th.join(gdbToBeUpdated,DOM_FeatAlt))
arcpy.CopyRows_management(os.path.join(DomainGDB,DOM_FeatLife),os.p
ath.join(gdbToBeUpdated,DOM_FeatLife))
arcpy.CopyRows_management(os.path.join(DomainGDB,DOM_ORIG),os.path.
join(gdbToBeUpdated,DOM_ORIG))
arcpy.CopyRows_management(os.path.join(DomainGDB,DOM_UnitCodeAlt),o
s.path.join(gdbToBeUpdated,DOM_UnitCodeAlt))
```

```

arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated,DOM_Change),'Code','Code',gdbToBeUpdated,DOM_Change,"Reason for feature change compared to previous version of dataset.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated,DOM_DataCert),'Code','Code',gdbToBeUpdated,DOM_DataCert,"Data Certification Status.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated,DOM_EDITED),'Code','Label',gdbToBeUpdated,DOM_EDITED,"Who last edited the feature. May not be the same as ORIGINATOR.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated,DOM_FeatAlt),'Code','Code',gdbToBeUpdated,DOM_FeatAlt,"How feature differs from NPS Land Resources Division dataset.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated,DOM_FeatLife),'Code','Code',gdbToBeUpdated,DOM_FeatLife,"Feature status.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated,DOM_ORIG),'Code','Label',gdbToBeUpdated,DOM_ORIG,"Who originally created the feature. May not be the same as EDITEDBY.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated,DOM_UnitCodeAlt),'Code','Code',gdbToBeUpdated,DOM_UnitCodeAlt,"How unit code differs from NPS Land Resources Division dataset.")
arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,pyToBeUpdated),"ChangeReasonFromPrecedingVersion",DOM_Change)
arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,pyToBeUpdated),"DataCertificationLevel",DOM_DataCert)
arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,pyToBeUpdated),"EDITEDBY",DOM_EDITED)
arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,pyToBeUpdated),"FeatureAlteration",DOM_FeatAlt)
arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,pyToBeUpdated),"FeatureLifecycle",DOM_FeatLife)
arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,pyToBeUpdated),"ORIGINATOR",DOM_ORIG)
arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,pyToBeUpdated),"UnitCodeAlteration",DOM_UnitCodeAlt)
arcpy.FeatureToPoint_management(pyToBeUpdated,ptToBeUpdated,"INSIDE")
arcpy.DeleteField_management(ptToBeUpdated,"ORIG_FID")
arcpy.CalculateField_management(ptToBeUpdated,"GEOMETRYID","IMD-UBParkPt-4326-\"!UnitCodeDisplay!\"-\"!IMDCreateDate!\",\"PYTHON_9.3")

```

Appendix E. Additional Processing Script for IMD Operational Boundaries

Additional processing script to run to create posted output for IMD Operational Boundaries.

```
import arcpy
import os, time

gdbToBeUpdated = r'' # this value is the file geodatabase of the
I&M Operational Boundaries polygon and point feature classes
pyToBeUpdated = r'' # this value is the wgs polygon feature class
within I&M Operational Boundaries
pyAlbers = r'' # this value is the albers polygon feature class
within I&M Operational Boundaries
pyServices = r'' # this value is the wgs Services polygon feature
class within I&M Operational Boundaries
ptToBeUpdated = r'' # this value is the wgs point feature class
within I&M Operational Boundaries
DomainGDB = r'' # Domain GDB: see cited reference (NPS IMD 2018f)

DOM_Change = 'DOM_ChangeReasonFromPrecedingVersion_NPSIMD2019'
DOM_DataCert = 'DOM_DataCertificationLevel_NPSIMD2019'
DOM_EDITED = 'DOM_EDITEDBY_NPSIMD2019'
DOM_FeatAlt = 'DOM_FeatureAlteration_NPSIMD2019'
DOM_FeatLife = 'DOM_FeatureLifecycle_NPSIMD2019'
DOM_ORIG = 'DOM_ORIGINATOR_NPSIMD2019'
DOM_UnitCodeAlt = 'DOM_UnitCodeAlteration_NPSIMD2019'

Features = [pyToBeUpdated, pyAlbers, pyServices, ptToBeUpdated]
Domains = [DOM_Change, DOM_DataCert, DOM_EDITED, DOM_FeatAlt,
DOM_FeatLife, DOM_ORIG, DOM_UnitCodeAlt]

for feat in Features:

arcpy.RemoveDomainFromField_management(feat,"DataCertificationLevel
")
    arcpy.RemoveDomainFromField_management(feat,"FeatureQualifier")
    arcpy.RemoveDomainFromField_management(feat,"Feature_Source")
    arcpy.RemoveDomainFromField_management(feat,"Lifecycle")

arcpy.RemoveDomainFromField_management(feat,"UnitCodeQualifier")
    arcpy.RemoveDomainFromField_management(feat,"ChangeReason")
    arcpy.RemoveDomainFromField_management(feat,"RedactReason")
```

```

arcpy.DeleteDomain_management(gdbToBeUpdated,"ChangeReason_Domain")
arcpy.DeleteDomain_management(gdbToBeUpdated,"DataCertificationLevel_Domain")
arcpy.DeleteDomain_management(gdbToBeUpdated,"FeatureQualifier_Domain")
arcpy.DeleteDomain_management(gdbToBeUpdated,"FeatureSource_Domain")
arcpy.DeleteDomain_management(gdbToBeUpdated,"Lifecycle_Domain")
arcpy.DeleteDomain_management(gdbToBeUpdated,"RedactReason_Domain")
arcpy.DeleteDomain_management(gdbToBeUpdated,"UnitCodeQualifier_Domain")

```

```

arcpy.Delete_management(os.path.join(gdbToBeUpdated,"ChangeReason_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"DataCertificationLevel_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"FeatureQualifier_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"FeatureSource_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"Lifecycle_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"RedactReason_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"UnitCodeQualifier_Domain"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"Lifecycle"))
arcpy.Delete_management(os.path.join(gdbToBeUpdated,"Feature_Source"))

```

```

for feat in Features:

```

```

    arcpy.AlterField_management(feat,"UNIT_CODE","UNITCODE",'UNITCODE',"TEXT",16,"NULLABLE",'None')

```

```

    arcpy.AlterField_management(feat,"UNIT_NAME","UNITNAME","UNITNAME","TEXT",254,"NULLABLE",'None')

```

```

    arcpy.AlterField_management(feat,"DATE_EDIT","EDITDATE","EDITDATE","DATE","", "NULLABLE", "None")

```

```
arcpy.AlterField_management(feats, "Unit_Code_Display", "UnitCodeDisplay", "UnitCodeDisplay", "TEXT", 25, "NULLABLE", 'None')
```

```
arcpy.AlterField_management(feats, "Feature_Source", "ORIGINATOR", "ORIGINATOR", "TEXT", 100, "NULLABLE", 'None')
```

```
arcpy.AlterField_management(feats, "Source_Metadata", "OriginatorMetadata", "OriginatorMetadata", "TEXT", 150, "NULLABLE", 'None')
```

```
arcpy.AlterField_management(feats, "Identifier", "GEOMETRYID", "GEOMETRYID", "TEXT", 250, "NULLABLE", 'None')
```

```
arcpy.AlterField_management(feats, "CreateDate", "IMDCreatesDate", "IMDCreatesDate", "TEXT", 10, "NULLABLE", 'None')
```

```
arcpy.AlterField_management(feats, "DSLink", "FeatureMetadata", "FeatureMetadata", "TEXT", 150, "NULLABLE", 'None')
```

```
arcpy.AlterField_management(feats, "Lifecycle", "FeatureLifecycle", "FeatureLifecycle", "TEXT", 15, "NULLABLE", 'None')
```

```
arcpy.AlterField_management(feats, "Network_Code", "NetworkCode", "NetworkCode", "TEXT", 4, "NULLABLE")
```

```
arcpy.AddField_management(feats, "ChangeReasonFromPrecedingVersion", "TEXT", '', '', 50, "ChangeReasonFromPrecedingVersion", "NULLABLE", "NON_REQUIRED")
```

```
arcpy.CalculateField_management(feats, "ChangeReasonFromPrecedingVersion", '!ChangeReason!', "PYTHON_9.3")
```

```
arcpy.DeleteField_management(feats, "ChangeReason")
```

```
arcpy.DeleteField_management(feats, "Full_Name")
```

```
arcpy.DeleteField_management(feats, "RedactReason")
```

```
arcpy.AddField_management(feats, "EDITEDBY", "TEXT", '', '', 100, "EDITEDBY", "NULLABLE", "NON_REQUIRED")
```

```

arcpy.AlterField_management(feats,"UnitCodeQualifier","UnitCodeAlter
ation","UnitCodeAlteration","TEXT",10,'NULLABLE','None')

arcpy.AlterField_management(feats,"FeatureQualifier","FeatureAlterat
ion","FeatureAlteration","TEXT",10,"NULLABLE","NON_REQUIRED")

arcpy.JoinField_management(feats,"UnitCodeAlteration",os.path.join(D
omainGDB,DOM_UnitCodeAlt),"Code","UnitCodeAlteration_Label")

arcpy.JoinField_management(feats,"FeatureAlteration",os.path.join(Do
mainGDB,DOM_FeatAlt),"Code","FeatureAlteration_Label")

    arcpy.MakeFeatureLayer_management(feats,"py_temp")

arcpy.SelectLayerByAttribute_management("py_temp","NEW_SELECTION","
ChangeReasonFromPrecedingVersion = 'None'")

arcpy.CalculateField_management("py_temp","ChangeReasonFromPrecedin
gVersion",' "NC"', "PYTHON_9.3")

arcpy.SelectLayerByAttribute_management("py_temp","SWITCH_SELECTION
")

arcpy.CalculateField_management("py_temp","ChangeReasonFromPrecedin
gVersion",' "LRD"', "PYTHON_9.3")

##
arcpy.SelectLayerByAttribute_management("py_temp","NEW_SELECTION","
IMDCreateDate = '20181003'")
##
arcpy.CalculateField_management("py_temp","ChangeReasonFromPrecedin
gVersion",' "LRD"', "PYTHON_9.3")
##
arcpy.SelectLayerByAttribute_management("py_temp","CLEAR_SELECTION"
)

arcpy.JoinField_management(feats,"ChangeReasonFromPrecedingVersion",
os.path.join(DomainGDB,DOM_Change),'Code',"ChangeReasonFromPrecedin
gVersion_Label")

```

```

arcpy.SelectLayerByAttribute_management("py_temp", "CLEAR_SELECTION"
)

arcpy.CalculateField_management(feats, "EDITEDBY", "!ORIGINATOR!", "PYTHON_9.3")
    arcpy.DeleteField_management(feats, "ChangeReason_Label")
    arcpy.DeleteField_management(feats, "FeatureQualifier_Label")
    arcpy.DeleteField_management(feats, "UnitCodeQualifier_Label")

    arcpy.Delete_management("py_temp")

arcpy.CopyRows_management(os.path.join(DomainGDB, DOM_Change), os.path.join(gdbToBeUpdated, DOM_Change))
arcpy.CopyRows_management(os.path.join(DomainGDB, DOM_DataCert), os.path.join(gdbToBeUpdated, DOM_DataCert))
arcpy.CopyRows_management(os.path.join(DomainGDB, DOM_EDITED), os.path.join(gdbToBeUpdated, DOM_EDITED))
arcpy.CopyRows_management(os.path.join(DomainGDB, DOM_FeatAlt), os.path.join(gdbToBeUpdated, DOM_FeatAlt))
arcpy.CopyRows_management(os.path.join(DomainGDB, DOM_FeatLife), os.path.join(gdbToBeUpdated, DOM_FeatLife))
arcpy.CopyRows_management(os.path.join(DomainGDB, DOM_ORIG), os.path.join(gdbToBeUpdated, DOM_ORIG))
arcpy.CopyRows_management(os.path.join(DomainGDB, DOM_UnitCodeAlt), os.path.join(gdbToBeUpdated, DOM_UnitCodeAlt))

arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated, DOM_Change), 'Code', 'Code', gdbToBeUpdated, DOM_Change, "Reason for feature change compared to previous version of dataset.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated, DOM_DataCert), 'Code', 'Code', gdbToBeUpdated, DOM_DataCert, "Data Certification Status.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated, DOM_EDITED), 'Code', 'Label', gdbToBeUpdated, DOM_EDITED, "Who last edited the feature. May not be the same as ORIGINATOR.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated, DOM_FeatAlt), 'Code', 'Code', gdbToBeUpdated, DOM_FeatAlt, "How feature differs from NPS Land Resources Division dataset.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated, DOM_FeatLife), 'Code', 'Code', gdbToBeUpdated, DOM_FeatLife, "Feature status.")

```

```

arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated,DOM_ORIG
),'Code','Label',gdbToBeUpdated,DOM_ORIG,"Who originally created
the feature. May not be the same as EDITEDBY.")
arcpy.TableToDomain_management(os.path.join(gdbToBeUpdated,DOM_Unit
CodeAlt),'Code','Code',gdbToBeUpdated,DOM_UnitCodeAlt,"How unit
code differs from NPS Land Resources Division dataset.")

for feat in Features:

arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,fe
at),"ChangeReasonFromPrecedingVersion",DOM_Change)

arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,fe
at),"DataCertificationLevel",DOM_DataCert)

arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,fe
at),"EDITEDBY",DOM_EDITED)

arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,fe
at),"FeatureAlteration",DOM_FeatAlt)

arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,fe
at),"FeatureLifecycle",DOM_FeatLife)

arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,fe
at),"ORIGINATOR",DOM_ORIG)

arcpy.AssignDomainToField_management(os.path.join(gdbToBeUpdated,fe
at),"UnitCodeAlteration",DOM_UnitCodeAlt)

```

Appendix F. Session and Version Information

Windows 10 Enterprise

Version 1803

Installed on 9/26/2018

OS build 17134.984

ArcMap/Catalog 10.6.1.9270

Python 2.7.3 64-bit

Pyscripter 3.6.0.0 x64

The Department of the Interior protects and manages the nation's natural resources and cultural heritage; provides scientific and other information about those resources; and honors its special responsibilities to American Indians, Alaska Natives, and affiliated Island Communities.

NPS 909/166612, January 2020

National Park Service
U.S. Department of the Interior



[Natural Resource Stewardship and Science](#)

1201 Oakridge Drive, Suite 150
Fort Collins, CO 80525