



crds Documentation

Release 0.1

STScI

September 14, 2012

CONTENTS

1	Installation	1
1.1	Getting the Source Code	1
1.2	Setting up your Environment	1
1.3	Run the Install Script	2
1.4	Test the installation	2
1.5	Package Overview	2
1.6	Dependencies	3
2	Top Level Use	5
2.1	crds.getreferences()	5
2.2	crds.get_default_context()	6
3	Non-Networked Use	7
3.1	Overview of Features	7
3.2	Important Modules	7
3.3	Basic Operations on Mappings	8
3.4	Certifying Files	10
3.5	Mapping Checksums	11
3.6	Which Mappings Use this File?	11
3.7	Finding Matches for a Reference in a Context	12
4	About Mappings	13
4.1	Naming	13
4.2	Basic Structure	14
4.3	Pipeline Mappings (.pmap)	14
4.4	Instrument Mappings (.imap)	14
4.5	Reference Mappings (.rmap)	15
4.6	Selectors	16
5	Command Line Tools	21
5.1	Specifying Files	21
5.2	crds.certify	21
5.3	crds.diff	22
5.4	crds.uses	22
5.5	crds.matches	22
5.6	crds.sync	23
6	Using the CRDS Web Site	25
6.1	Public Functions	25
6.2	Private Functions	30

INSTALLATION

1.1 Getting the Source Code

At this stage of development, installing CRDS is accomplished by checking CRDS source code out from subversion:

```
% svn co https://subversion.assembla.com/svn/crds/trunk crds
% cd crds
```

1.2 Setting up your Environment

The CRDS checkout has a template file for the C-shell which defines environment variables, `env.csh`. For JWST development, there are reasonable defaults for everything so you may not need to set these at all. For HST, at a minimum `CRDS_SERVER_URL` must be defined.

1.2.1 Basic Environment

- **CRDS_PATH** defines a common directory tree where CRDS reference files and mappings are stored. `CRDS_PATH` defaults to `"/crds"`. Mappings are stored in `${CRDS_PATH}/mappings/[hstljwst]`. Reference files are stored in `${CRDS_PATH}/references/[hstljwst]`.
- **CRDS_SERVER_URL** defines the base URL for accessing CRDS network services. `CRDS_SERVER_URL` defaults to the jwst test server.
- **CRDS_VERBOSITY** enables output of CRDS debug messages. Set to an integer, nominally 50. Higher values output more information, lower values less information.

1.2.2 Advanced Environment

- **CRDS_MAPPATH** can be used to override `CRDS_PATH` and define where only mapping files are stored. If mappings are pre-installed, the directory pointed to by `CRDS_MAPPATH` can be readonly. `CRDS_MAPPATH` defaults to `${CRDS_PATH}/mappings`.
- **CRDS_REFPATH** can be used to override `CRDS_PATH` and define where only reference files are stored. If references are pre-installed, the directory pointed to by `CRDS_REFPATH` can be readonly. `CRDS_REFPATH` defaults to `${CRDS_PATH}/references`.
- **CRDS_CFGPATH** can be used to override `CRDS_PATH` and define where only server configuration information is cached. The directory pointed to by `CRDS_CFGPATH` should be writable. If CRDS is running in server-less mode, this path is irrelevant. `CRDS_CFGPATH` defaults to `${CRDS_PATH}/config`.

- **CRDS_MODE** defines whether CRDS should compute best references using client software (local), server software (remote), or intelligently “fall up” to the server only when the client is deemed obsolete or the server cannot be reached (auto). The default is auto.
- **CRDS_CONTEXT** is an override naming the CRDS pipeline mapping (.pmap) used for computing best references. Ordinarily, CRDS will contact the server to determine the operational pipeline mapping. If the server cannot be reached, CRDS will look in CRDS_CFGPATH to determine the last pipeline context the server recommended. If there is no prior server info available in the cache, CRDS will fall-back to using the default pre-installed mappings, e.g. `jwt.pmap`. When CRDS_CONTEXT is set, CRDS will ignore server recommendations and availability and use the specified pipeline mapping. However, CRDS_CONTEXT will only be used when `context` was specified to `getreferences()` as `None`. If `context` was explicitly specified in a call to `getreferences()` and was not `None`, the specified context will override CRDS_CONTEXT. This enables the implementation of command line switches which supercede CRDS_CONTEXT.

Edit `env.csh` according to your preferences for where to put CRDS files. Then source `env.csh` to define the variables in your environment:

```
% source env.csh
```

1.3 Run the Install Script

CRDS is partitioned into 3-4 Python packages each of which has it's own `setup.py` script. To make things easier, the top level directory has a single “install” script which runs all the individual `setup.py` scripts for you:

```
% ./install
Installing lib
Installing client
Installing hst
Installing jwt
Installing tobs
final status 000000
```

1.4 Test the installation

CRDS client testing operates locally and does not require access to the server. Basic CRDS client testing can be done as follows:

```
% ./runtests
... copious test output...
```

```
-----
Ran 59 tests in 13.749s
```

```
OK
```

1.5 Package Overview

From the perspective of an end user, CRDS consists of 3 or more Python packages which implement different capabilities:

- **crds**
 - core package enabling local use and development of mappings and reference files.

- **crds.client**
 - network client library for interacting with the central CRDS server.
- **crds.hst**
 - observatory personality package for HST, with initial mappings for bootstrapping CRDS and defining how HST files are named, located, and certified.
- **crds.jwst**
 - analogous to crds.hst, for JWST.

1.6 Dependencies

CRDS was developed in and for an STSCI Python environment suitable for pipeline processing. CRDS requires these additional packages to be installed in your Python environment:

- numpy
- pyfits

For executing the unit tests (runtests) add:

- nose
- BeautifulSoup
- stsci.tools

For building documentation add:

- stsci.sphinxext

TOP LEVEL USE

This section describes the formal top level interfaces for CRDS intended as the main entry points for the calibration software or basic use. Functions at this level should be assumed to require network connectivity with the CRDS server.

To function correctly, these API calls may require the user to set the environment variables `CRDS_SERVER_URL` and `CRDS_PATH`. See the section on *Installation* for more details.

2.1 `crds.getreferences()`

Given dataset header containing parameters required to determine best references, and optionally a specific .pmap to use as the best references context, and optionally a list of the reference types for which reference files are to be determined, `getreferences()` will determine best references, cache them on the local file system, and return a mapping from reference types to reference file paths:

```
def getreferences(parameters, reftypes=None, context=None, ignore_cache=False,
                  observatory="jwst"):
```

"""Return the mapping from the requested 'reftypes' to their
corresponding best reference file paths appropriate for a dataset
described by 'parameters' with CRDS rules defined by 'context'::

`parameters` : A mapping of parameter names to parameter value
strings for parameters which define best reference file matches.

{ str : str, int, float, bool }

e.g. {

 'INSTRUME' : 'ACS',

 'CCDAMP' : 'ABCD',

 'CCDGAIN' : '2.0',

 ...

}

`reftypes` : A list of reference type names. For HST these are the keywords
used to record reference files in dataset headers. For JWST, these
are the identifiers which will appear in instrument contexts and
reference mappings.

e.g. ['darkfile', 'biasfile']

If `reftypes` is `None`, return all reference types defined by
the instrument mapping for the instrument specified in
'parameters'.

```
context : The name of the pipeline context mapping which should be
          used to define best reference lookup rules, or None. If
          'context' is None, use the latest operational pipeline mapping.

          str

          e.g. 'hst_0037.pmap'

ignore_cache : If True, download all required mappings and references
               from the CRDS server. If False, download only those files not
               already in the local caches.

observatory : The name of the observatory this query applies to, needed
              to support both 'hst' and 'jwst' from a single server.

Returns
-----
a mapping from reftypes to cached best reference file paths.

{ str : str }

e.g. {
      'biasfile' : '/path/to/file/hst_acs_biasfile_0042.fits',
      'darkfile' : '/path/to/file/hst_acs_darkfile_0056.fits',
    }
"""
```

2.2 crds.get_default_context()

get_default_context() returns the name of the pipeline mapping which is currently in operational use. When no The default context defines the matching rules used to determine best reference files for a given set of parameters:

```
def get_default_context():
    """Return the name of the latest pipeline mapping in use for processing
    files.

    Returns
    -----
    pipeline context name

    e.g. 'hst_0007.pmap'
    """
```

NON-NETWORKED USE

This section describes using the core crds package without access to the network. Using the crds package in isolation it is possible to develop and use new reference files and mappings. Note that a default install of CRDS will also include crds.client and crds.hst or crds.jwst. In particular, the observatory packages define how mappings are named, where they are placed, and how reference files are checked.

3.1 Overview of Features

Using the crds package it's possible to:

- Load and operate on rmaps
- Determine best reference files for a dataset
- Check mapping syntax and verify checksum
- Certify that a mapping and all it's dependencies exist and are valid
- Certify that a reference file meets important constraints
- Add checksums to mappings
- Determine the closure of mappings which reference a particular file.

3.2 Important Modules

There are really two important modules which anyone doing low-level and non- networked CRDS development will first be concerned with:

- **crds.rmap module**
 - defines classes which load and operate on mapping files
 - Mapping
 - PipelineContext (.pmap)
 - InstrumentContext (.imap),
 - ReferenceMapping (.rmap)
 - defines `get_cached_mapping()` function

- loads and caches a Mapping or subclass instances from files, typically this is a recursive process loading pipeline or instrument contexts as well as all associated reference mappings.
- this *cache* is an object cache to speed up access to mappings, not the file *cache* used by `crds.client` to avoid repeated network file transfers.

- **crds.selectors module**

- **defines classes implementing best reference logic**

- MatchSelector
 - UseAfterSelector
 - Other experimental Selector classes

3.3 Basic Operations on Mappings

3.3.1 Loading Rmaps

Perhaps the most fundamental thing you can do with a CRDS mapping is create an active object version by loading the file:

```
% python
>>> import crds.rmap as rmap
>>> hst = rmap.load_mapping("hst.pmap")
```

The `load_mapping()` function will take any mapping and instantiate it and all of its child mappings into various nested Mapping subclasses: PipelineContext, InstrumentContext, or ReferenceMapping.

Loading an rmap implicitly screens it for invalid syntax and requires that the rmap's checksum (`sha1sum`) be valid by default.

Since HST has on the order of 70 mappings, this is a fairly slow process requiring a couple seconds to execute. In order to speed up repeated access to the same Mapping, there's a mapping cache maintained by the rmap module and accessed like this:

```
>>> hst = rmap.get_cached_mapping("hst.pmap")
```

The behavior of the cached mapping is identical to the “loaded” mapping and subsequent calls are nearly instant.

3.3.2 Seeing Referenced Names

CRDS Mapping classes all know how to show you the files referenced by themselves and their descendents. The ACS instrument context has a reference mapping for each of its associated file kinds:

```
>>> acs = rmap.get_cached_mapping("hst_acs.imap")
>>> acs.mapping_names()
['hst_acs.imap',
 'hst_acs_idctab.rmap',
 'hst_acs_darkfile.rmap',
 'hst_acs_atodtab.rmap',
 'hst_acs_cfltfile.rmap',
 'hst_acs_spottab.rmap',
 'hst_acs_mlintab.rmap',
```

```
'hst_acs_dgeofile.rmap',
'hst_acs_bpixtab.rmap',
'hst_acs_oscntab.rmap',
'hst_acs_ccdtab.rmap',
'hst_acs_crrehtab.rmap',
'hst_acs_pfltfile.rmap',
'hst_acs_biasfile.rmap',
'hst_acs_mdrihtab.rmap']
```

The ACS atod reference mapping (rmap) refers to 4 different reference files:

```
>>> acs_atod = rmap.get_cached_mapping("hst_acs_atodtab.rmap")
>>> acs_atod.reference_names()
['j4d1435hj_a2d.fits',
'kcb1734hj_a2d.fits',
'kcb1734ij_a2d.fits',
't3n1116mj_a2d.fits']
```

3.3.3 Computing Best References

The primary function of CRDS is the computation of best reference files based upon a dictionary of dataset metadata. Hence, both an InstrumentContext and a ReferenceMapping can meaningfully return the best references for a dataset based upon a parameter dictionary. It's possible to define a header as any Python dictionary provided you have sufficient knowledge of the parameters:

```
>>> hdr = { ... what matters most ... }
```

On the other hand, if your dataset is a FITS file and you want to do something quick and dirty, you can ask CRDS what dataset metadata may matter for determining best references:

```
>>> hdr = acs.get_minimum_header("test_data/j8bt05njq_raw.fits")
{'CCDAMP': 'C',
'CCDGAIN': '2.0',
'DATE-OBS': '2002-04-13',
'DETECTOR': 'HRC',
'FILTER1': 'F555W',
'FILTER2': 'CLEAR2S',
'FW1OFFST': '0.0',
'FW2OFFST': '0.0',
'FWSOFFST': '0.0',
'LTV1': '19.0',
'LTV2': '0.0',
'NAXIS1': '1062.0',
'NAXIS2': '1044.0',
'OBSERVE': 'IMAGING',
'TIME-OBS': '18:16:35'}
```

Here I say *may matter* because CRDS is currently dumb about specific instrument configurations and is returning metadata about filekinds which may be inappropriate.

Once you have your dataset parameters, you can ask an InstrumentContext for the best references for *all* filekinds for that instrument:

```
>>> acs.get_best_references(hdr)

{'atodtab': 'kcb1734ij_a2d.fits',
'biasfile': 'm4r1753rj_bia.fits', 'bpixtab': 'm8r09169j_bpx.fits', 'ccdtab': 'o1515069j_ccd.fits', 'cflt-
file': 'NOT FOUND n/a', 'crrehtab': 'n4e12510j_crr.fits', 'darkfile': 'n3o1059hj_drk.fits', 'dge-
```

```
ofile': 'o8u2214mj_dxy.fits', 'flshfile': 'NOT FOUND n/a', 'idctab': 'p7d1548qj_idc.fits', 'impht-  
tab': 'vbb18105j_imp.fits', 'mdriztab': 'ub215378j_mdz.fits', 'mlintab': 'NOT FOUND n/a', 'osentab':  
'm2j1057pj_osc.fits', 'pfltfile': 'o3u1448rj_pfl.fits', 'shadfile': 'kcb1734pj_shd.fits', 'spottab': 'NOT FOUND  
n/a'}
```

In the above results, FITS files are the recommended best references, while a value of “NOT FOUND n/a” indicates that no result was expected for the current instrument mode as defined in the header. Other values of “NOT FOUND xxx” include an error message xxx which hints at why no result was found, such as an invalid dataset parameter value or simply a matching failure.

You can ask a ReferenceMapping for the best reference for single the filekind it manages:

```
>>> acs_atod.get_best_ref(hdr)  
>>> 'kcb1734ij_a2d.fits'
```

Often it is convenient to simply refer to a pipeline/observatory context file, and hence PipelineContext can also return the best references for a dataset, but this is really just shorthand for returning the best references for the instrument of that dataset:

```
>>> hdr = hst.get_minimum_header("test_data/j8bt05njq_raw.fits")  
>>> hst.get_best_references(hdr)  
... for this hdr, same as acs.get_best_references(hdr) ...
```

Here it is critical to call `get_minimum_header` on the pipeline context, `hst`, because this will make it include the “INSTRUME” parameter needed to choose the ACS instrument.

3.4 Certifying Files

CRDS has a module which will certify that a mapping or reference file is valid, for some limited definition of *valid*. By design only valid files can be submitted to the CRDS server and archive.

3.4.1 Certifying Mappings

For Mappings, `crds.certify` will ensure that:

- the mapping and it’s descendents successfully load
- the mapping checksum is valid
- the mapping does not contain hostile code
- the mapping defines certain generic parameters
- references required by the mapping exist on the local file system

You can check the validity of your mapping or reference file like this:

```
% python -m crds.certify /where/it/really/is/hst_acs_my_masterpiece.rmap  
0 errors  
0 warnings  
0 infos
```

By default, running `certify` on a mapping *does not* verify that the required reference files are valid, only that they exist.

Later versions of CRDS may have additional semantic checks on the correctness of Mappings but these are not yet implemented and hence fall to the developer to verify in some other fashion.

3.4.2 Certifying Reference Files

For reference files `certify` has better semantic checks. For reference files, `crds.certify` currently ensures that:

- the FITS format is valid
- critical reference file header parameters have acceptable values

You can certify reference files the same way as mappings, like this:

```
% python -m crds.certify /where/it/is/my_reference_file.fits
0 errors
0 warnings
0 infos
```

3.5 Mapping Checksums

CRDS mappings contain `sha1sum` checksums over the entire contents of the mapping, with the exception of the checksum itself. When a CRDS Mapping of any kind is loaded, the checksum is transparently verified to ensure that the Mapping contents are intact.

3.5.1 Ignoring Checksums!

Ordinarily, during pipeline operations, ignoring checksums should not be done. Ironically though, the first thing you may want to do as a developer is ignore the checksum while you load a mapping you've edited:

```
>>> pipeline = rmap.load_mapping("hst.pmap", ignore_checksum=True)
```

3.5.2 Adding Checksums

Once you've finished your masterpiece `ReferenceMapping`, it can be sealed with a checksum like this:

```
% python -m crds.checksum /where/it/really/is/hst_acs_my_masterpiece.rmap
```

3.6 Which Mappings Use this File?

Particularly in legacy contexts, such as HST, reference file names can be rather cryptic. Further, by design CRDS will have a complex set of fluid and versioned mappings. Hence it may become rather difficult for a human to discern which mappings refer to a particular mapping or reference file. CRDS has the `crds.uses` module to help answer this question:

```
% python -m crds.uses hst kcb1734ij_a2d.fits
hst.pmap
hst_acs.imap
hst_acs_atodtab.rmap
```

The first parameter indicates the observatory for which files should be considered. Additional parameters specify mapping or reference files which are used. The printed result consists of those mappings which directly or indirectly refer to the used files.

Note that the above results represent the highly simplified context of the current HST prototype, prior to the introduction of mapping evolution and version numbering. In practice, each of the above files might include several numbered versions, and some versions of the above files might not require `kcb1734ij_a2d.fits`.

crds.us knows about only the mappings cached locally. Hence the official CRDS server will have a more definitive answer than someone's development machine. The CRDS web site has a link for running crds.us over all known "official" mappings. crds.us is especially applicable for understanding the implications of blacklisting a particular file; when a file is blacklisted, all files indicated by crds.us are also blacklisted.

3.7 Finding Matches for a Reference in a Context

Given a particular context and reference file name, CRDS can also determine all possible matches for the reference within that context:

```
% python -m crds.matches hst.pmap kcb1734ij_a2d.fits
```

[illegible]

What is printed out is a sequence of match tuples, with each tuple nominally consisting of three parts:

```
(pmap_imap_rmap_path, match, use_after)
```

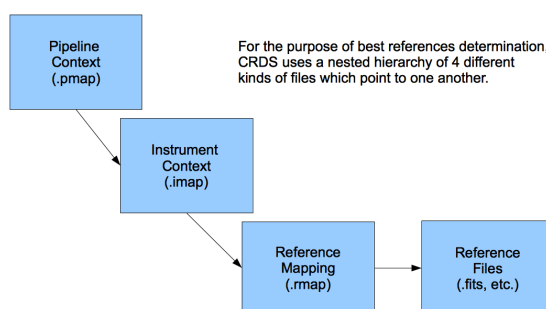
Each part in turn consists of nested tuples of the form:

```
(parkey, value)
```


ABOUT MAPPINGS

CRDS mappings are organized in a 3 tier hierarchy: pipeline (.pmap), instrument (.imap), and reference (.rmap). Based on dataset parameters, the pipeline context is used to select an instrument mapping, the instrument mapping is used to select a reference mapping, and finally the reference mapping is used to select a reference file.

CRDS mappings are written in a subset of Python and given the proper global definitions can be parsed directly by the Python interpreter. Nothing precludes writing a parser for CRDS mappings in some other language.



4.1 Naming

The CRDS HST mapping prototypes which are generated from information scraped from the CDBS web site are named with the forms:

```
<observatory> .pmap                .e.g. hst.pmap
<observatory> _ <instrument> .imap   .e.g. hst_acs.imap
<observatory> _ <instrument> _ <filekind> .rmap .e.g. hst_acs_darkfile.rmap
```

The names of subsequent derived mappings include a version number:

```
<observatory> _ <version> .pmap                .e.g. hst_00001.pmap
<observatory> _ <instrument> _ <version> .imap   .e.g. hst_acs_00047.imap
<observatory> _ <instrument> _ <filekind> _ <version> .rmap .e.g. hst_acs_darkfile_00012.rmap
```

4.2 Basic Structure

All mappings have the same basic structure consisting of a “header” section followed by a “selector” section. The header provides meta data describing the mapping, while the selector provides matching rules used to look up the results of the mapping. A critical field in the mapping header is the “parkey” field which names the dataset header parameters which are used by the selector to do its lookup.

4.3 Pipeline Mappings (.pmap)

A sample pipeline mapping for HST looks like:

```
header = {
    'name' : 'hst.pmap',
    'derived_from' : 'created by hand 12-23-2011',
    'mapping' : 'PIPELINE',
    'observatory' : 'HST',
    'parkey' : ('INSTRUME',),
    'description' : 'Initially generated on 12-23-2011',
    'shasum' : 'e2c6392fd2731df1e8d933bd990f3fd313a813db',
}

selector = {
    'ACS' : 'hst_acs.imap',
    'COS' : 'hst_cos.imap',
    'NICMOS' : 'hst_nicmos.imap',
    'STIS' : 'hst_stis.imap',
    'WFC3' : 'hst_wfc3.imap',
    'WFPC2' : 'hst_wfpc2.imap',
}
```

A pipeline mapping matches the dataset “INSTRUME” header keyword against its selector to look up an instrument mapping file.

4.4 Instrument Mappings (.imap)

A sample instrument mapping for HST’s COS instrument looks like:

```
header = {
    'derived_from' : 'scraped 2011-12-23 11:57:10',
    'description' : 'Initially generated on 2011-12-23 11:57:10',
    'instrument' : 'COS',
    'mapping' : 'INSTRUMENT',
    'name' : 'hst_cos.imap',
    'observatory' : 'HST',
    'parkey' : ('REFTYPE',),
    'shasum' : '800fb1567cb5bed4031402c7396aeb86c5e1db61',
    'source_url' : 'http://www.stsci.edu/hst/observatory/cdbs/SIfileInfo/COS/reftablequeryindex',
}

selector = {
    'badttab' : 'hst_cos_badttab.rmap',
    'bpixtab' : 'hst_cos_bpixtab.rmap',
    'brftab' : 'hst_cos_brftab.rmap',
    'brsttab' : 'hst_cos_brsttab.rmap',
}
```

```

    'deadtab' : 'hst_cos_deadtab.rmap',
    'disptab' : 'hst_cos_disptab.rmap',
    'flatfile' : 'hst_cos_flatfile.rmap',
    'flxstab' : 'hst_cos_fluxstab.rmap',
    'geofile' : 'hst_cos_geofile.rmap',
    'lamptab' : 'hst_cos_lamptab.rmap',
    'phatab' : 'hst_cos_phatab.rmap',
    'spwcstab' : 'hst_cos_spwcstab.rmap',
    'tdstab' : 'hst_cos_tdstab.rmap',
    'wcptab' : 'hst_cos_wcptab.rmap',
    'xtractab' : 'hst_cos_xtractab.rmap',
}

```

Instrument mappings match the desired reference file type against the reference mapping which can be used to determine a best reference recommendation for a particular dataset. An instrument mapping lists all possible reference types for all modes of the instrument, some of which may not be appropriate for a particular mode. The selector key of an instrument mapping is the value of a reference file header keyword “REFTYPE”, and is the name of the dataset header keyword which will record the best reference selection.

4.5 Reference Mappings (.rmap)

A sample reference mapping for HST COS DEADTAB looks like:

```

header = {
    'derived_from' : 'scraped 2011-12-23 11:54:56',
    'description' : 'Initially generated on 2011-12-23 11:54:56',
    'filekind' : 'DEADTAB',
    'instrument' : 'COS',
    'mapping' : 'REFERENCE',
    'name' : 'hst_cos_deadtab.rmap',
    'observatory' : 'HST',
    'parkey' : (('DETECTOR',), ('DATE-OBS', 'TIME-OBS')),
    'shasum' : 'e27984a6441d8aaa7cd28ead2267a6be4c3a153b',
}

selector = Match({
    ('FUV',) : UseAfter({
        '1996-10-01 00:00:00' : 's7g1700gl_dead.fits',
    }),
    ('NUV',) : UseAfter({
        '1996-10-01 00:00:00' : 's7g1700ql_dead.fits',
    }),
})

```

Reference mapping selectors are constructed as a nested hierarchy of selection operators which match against various dataset header keywords.

4.5.1 Parkeys

For reference mappings, the header “parkey” field is a tuple of tuples. Each stage of the nested selector consumes the next tuple of header keys. For the example above, the Match operator matches against the value of the dataset keyword “DETECTOR”. Based on that match, the selected UseAfter operator matches against the dataset’s “DATE-OBS” and “TIME-OBS” keywords to lookup the name of a reference file.

4.6 Selectors

All the CRDS selection operators are written to select either a filename *or* a nested operator. In the case of HST, the Match operator locates a nested UseAfter operator which in turn locates the reference file.

4.6.1 Match

Based on a dataset's header values, Match locates the match tuple which best matches the dataset. Conceptually this is a dictionary lookup. In actuality, CRDS processes each match parameter in succession, at each step eliminating match candidates that cannot possibly match.

Parameter Tuples and Simple Matches

The CRDS Match operator typically matches a dataset header against a tuple which defines multiple parameter values whose names are specified in the rmap header parkey:

```
("UVIS", "F122LP") : 'some_file_or_nested_selection'
```

Alternately, for simple use cases the Match operator can match against single strings, which is a simplified syntax for a 1-tuple:

```
'UVIS' : 'some_file_or_nested_selection'  
( 'UVIS', ) : 'this_is_the_equivalent_one_tuple'
```

Single Parameter Values

Each value within the match tuples of a Match operator can be an expression in its own right. There are a number of special values associated with each match expression: Ors **|**, Wildcards *****, Regular Expressions **()**, Literals **{ }**, Relational, between, N/A, and Substitutions.

Or |

Many CRDS match expressions consist of a series of match patterns separated by vertical bars. The vertical bar is read as “or” and means that a match occurs if either pattern matches that dataset header. For example, the expression:

```
("either_this|that", "1|2|3") : "some_file.fits"
```

will match:

```
("either_this", "2")
```

and also:

```
("that", "1")
```

Wild Cards *

By default, ***** is interpreted in CRDS as a glob pattern, much like UNIX shell file name matching. ***** matches any sequence of characters. The expression:

```
("F*122",) : "some_file.fits"
```

will match any value starting with “F” and ending with “122”.

Regular Expressions

CRDS can match on true regular expressions. A true regular expression match is triggered by bracketing the match in parentheses ():

```
("(^F[^\13]22$)",) : "some_file.fits"
```

The above corresponds to matching the regular expression “`^F[^\13]22$`” (note that the bracketing parentheses within the string are removed.) Regular expression syntax is explained in the Python documentation for the `re` module. The above expression will match values starting with “F”, followed by any character which is not “1” or “3” followed by “22”.

Literal Expressions

A literal expression is bracketed with curly braces {} and is matched without any interpretation whatsoever. Hence, special characters like `*` or `|` are interpreted literally rather than as `ors` or wildcards. The expression:

```
("{F|*G}",) : "some_file.fits"
```

matches the value “F*G” as opposed to “F” or anything ending with “G”.

Relational Expressions

Relational expressions are bracketed by the pound character `#`. Relational expressions do numerical comparisons on the header value to determine a match. Relational expressions have implicit variables and support the operators:

```
> >= < <= == and or
```

The expression:

```
("# >1 and <37 #",) : "some_file.fits"
```

will match any number greater than 1 and less than 37.

Between

A special relational operator “between” is used to simply express a range of numbers `>=` to the lower bound and `<` the upper bound, similar to Python slicing:

```
("between 1 47",) : "some_file.fits"
```

will match any number greater than or equal to 1 and less than 47. This is equivalent to:

```
("# >=1 and <47 #",) : "some_file.fits"
```

Note that “between” matches sensibly stack into a complete range. The expressions:

```
("between 1 47",) : "some_file.fits"  
("between 47 90", ) : "another_file.fits"
```

provide complete coverage for the range between 1 and 90.

N/A

Some rmaps have match tuple values of “N/A”, or Not Applicable. A value of N/A is matched as a special version of “*”, matching anything, but not affecting the “weight” of the match.

```
('HRC', 'N/A') : “some_file.fits”
```

There are a couple uses for N/A parameters. First, sometimes a parameter is irrelevant in the context of the other parameters. So for an rmap which covers multiple instrument modes, a parameter may not apply to all modes. Second, sometimes a parameter is relevant to custom lookup code, but is not used by the match directly. In this second case, the “N/A” parameter may be used by custom header preconditioning code to assist in mutating the other parameter values that *are* used in the match.

Substitution Parameters

Substitution parameters are short hand notation which eliminate the need to duplicate rmap rules. In order to support WFC3 biasfile conventions, CRDS rmaps permit the definition of meta-match-values which correspond to a set of actual dataset header values. For instance, when an rmap header contains a “substitutions” field like this:

```
'substitutions' : {  
  'CCDAMP' : {  
    'G280_AMPS' : ('ABCD', 'A', 'B', 'C', 'D', 'AC', 'AD', 'BC', 'BD'),  
  },  
},
```

then a match tuple line like the following could be written:

```
('UVIS', 'G280_AMPS', '1.5', '1.0', '1.0', 'G280-REF', 'T') : UseAfter({
```

Here the value of G280_AMPS works like this: first, reference files listed under that match tuple define CCDAMP=G280_AMPS. Second, datasets which should use those references define CCDAMP to a particular amplifier configuration, .e.g. ABCD. Hence, the reference file specifies a set of applicable amplifier configurations, while the dataset specifies a particular configuration. CRDS automatically expands substitutions into equivalent sets of match rules.

Match Weighting

Because of the presence of special values like regular expressions, CRDS uses a winnowing match algorithm which works on a parameter-by-parameter basis by discarding match tuples which cannot possibly match. After examining all parameters, CRDS is left with a list of candidate matches.

For each literal, *, or regular expression parameter that matched, CRDS increases its sense of the goodness of the match by 1. For each N/A that was ignored, CRDS doesn't change the weight of the match. The highest ranked match is the one CRDS chooses as best. When more than one match tuple has the same highest rank, we call this an “ambiguous” match. Ambiguous matches will either be merged, or treated as errors/exceptions that cause the match to fail. Talk about ambiguity.

For the initial HST rmaps, there are a number of match cases which overlap, creating the potential for ambiguous matches by actual datasets. For HST, all of the match cases refer to nested UseAfter selectors. A working approach for handling ambiguities here is to merge the two or more equal weighted UseAfter lists into a single combined UseAfter which is then searched.

The ultimate goal of CRDS is to produce clear non-overlapping rules. However, since the initial rmaps are generated from historical mission data in CDBS, there are eccentricities which need to be accommodated by merging or eventually addressed by human beings who will simplify the rules by hand.

4.6.2 UseAfter

The UseAfter selector matches an ordered sequence of date time values to corresponding reference filenames. UseAfter finds the greatest date-time which is less than or equal to (\leq) EXPSTART of a dataset. Unlike reference file and dataset timestamp values, all CRDS rmaps represent times in the single format shown in the rmap example below:

```
selector = Match({
  ('HRC',) : UseAfter({
    '1991-01-01 00:00:00' : 'j4d1435hj_a2d.fits',
    '1992-01-01 00:00:00' : 'kcb1734ij_a2d.fits',
  }),
  ('WFC',) : UseAfter({
    '1991-01-01 00:00:00' : 'kcb1734hj_a2d.fits',
    '2008-01-01 00:00:00' : 't3n1116mj_a2d.fits',
  }),
})
```

In the above mapping, when the detector is HRC, if the dataset's date/time is before 1991-01-01, there is no match. If the date/time is between 1991-01-01 and 1992-01-01, the reference file 'j4d1435hj_a2d.fits' is matched. If the dataset date/time is 1992-01-01 or after, the recommended reference file is 'kcb1734ij_a2d.fits'

4.6.3 SelectVersion

The SelectVersion() rmap operator uses a software version and various relations to make a selection:

```
selector = SelectVersion({
  '<3.1' : 'cref_flatfield_65.fits',
  '<5' : 'cref_flatfield_73.fits',
  'default' : 'cref_flatfield_123.fits',
})
```

While similar to relational expressions in Match(), SelectVersion() is dedicated, simpler, and more self-documenting. With the exception of default, versions are examined in sorted order.

4.6.4 ClosestTime

The ClosestTime() rmap operator does a lookup on a series of times and selects the closest time which either precedes or follows the given parameter value:

```
selector = ClosestTime({
  '2017-04-24 00:00:00' : "cref_flatfield_123.fits",
  '2018-02-01 00:00:00' : "cref_flatfield_222.fits",
  '2019-04-15 00:00:00' : "cref_flatfield_123.fits",
})
```

So a parameter of '2017-04-25 00:00:00' would select 'cref_flatfield_123.fits'.

4.6.5 GeometricallyNearest

The GeometricallyNearest() selector applies a distance relation between a numerical parameter and the match values. The match value which is closest to the supplied parameter is chosen:

```
selector = GeomtricallyNearest({
  1.2 : "cref_flatfield_120.fits",
  1.5 : "cref_flatfield_124.fits",
})
```

```
    5.0 : "cref_flatfield_137.fits",  
  })
```

In this case, a value of 1.3 would match 'cref_flatfield_120.fits'.

4.6.6 Bracket

The Bracket() selector is unusual because it returns the pair of selections which enclose the supplied parameter value:

```
selector = Bracket({  
    1.2: "cref_flatfield_120.fits",  
    1.5: "cref_flatfield_124.fits",  
    5.0: "cref_flatfield_137.fits",  
  })
```

Here, a parameter value of 1.3 returns the value:

```
('cref_flatfield_120.fits', 'cref_flatfield_124.fits')
```


COMMAND LINE TOOLS

Using the command line tools requires a local installation of the CRDS library. Some of the command line tools also interact with the CRDS server in order to implement their functionality.

5.1 Specifying Files

The command line tools operate on CRDS reference and mapping files in various ways. To specify a file in your local CRDS file cache, as defined by `CRDS_PATH`, use no path on the file:

```
% python -m crds.diff hst.pmap hst_0001.pmap # assumes paths in CRDS cache
```

To specify a particular file which is not located in your cache, give at least a relative path to the file, `./` will do:

```
% python -m crds.diff /some/path/hst.pmap ./hst_0002.pmap # uses given paths
```

5.2 `crds.certify`

`crds.certify` checks a reference or mapping file against constraints on legal matching parameter values. For reference files, `crds.certify` also performs checks of the FITS format and when given a context, and will compare the given file against the file it replaces looking for new or missing table rows.

`crds.certify --help` yields:

```
Usage: certify.py [options] <inpaths...>
```

Options:

<code>-h, --help</code>	show this help message and exit
<code>-d, --deep</code>	Certify reference files referred to by mappings have valid contents.
<code>-e, --exist</code>	Certify reference files referred to by mappings exist.
<code>-m, --mapping</code>	Ignore extensions, the files being certified are mappings.
<code>-p, --dump-provenance</code>	Print provenance keyword values.
<code>-t TRAP_EXCEPTIONS, --trap-exceptions=TRAP_EXCEPTIONS</code>	Capture exceptions at level: pmap, imap, rmap, selector, debug, none
<code>-x CONTEXT, --context=CONTEXT</code>	Pipeline context defining replacement reference.
<code>-V VERBOSITY, --verbose=VERBOSITY</code>	Set verbosity level.

`crds.certify` is invoked as, e.g.:

```
% python -m crds.certify --context=hst_0027.pmap    some_reference.fits
% python -m crds.certify hst.pmap
```

Invoking `crds.certify` on a context mapping recursively certifies all sub-mappings.

5.3 `crds.diff`

`crds.diff` compares two reference or mapping files and reports differences. For references `crds.diff` is currently a thin wrapper around `fitsdiff` but may expand.

For CRDS mappings `crds.diff` performs a recursive logical difference which shows the full match path to each bottom level change. `crds.diff --help` yields:

```
Usage: diff.py [options] <file1> <file2>
```

```
    Appropriately difference CRDS mapping or reference files.
```

Options:

```
-h, --help            show this help message and exit
-J, --jwst            Locate files using JWST naming conventions.
-H, --hst             Locate files using HST naming conventions.
-V VERBOSITY, --verbose=VERBOSITY
                        Set verbosity level.
```

For standard CRDS filenames, `crds.diff` can guess the observatory. For non-standard names, the observatory needs to be specified. `crds.diff` can be invoked like:

```
% python -m crds.diff    jwst_nircam_dark_0010.fits    jwst_nircam_dark_0011.fits

% python -m crds.diff    jwst_0001.pmap    jwst_0002.pmap
(('hst.pmap', 'hst_0004.pmap'), ('hst_acs.imap', 'hst_acs_0004.imap'), ('hst_acs_darkfile.rmap', 'hst_acs_0004_darkfile.rmap'))
```

5.4 `crds.uses`

`crds.uses` searches the files in the local cache for mappings which refer to the specified files. Since the **local cache** is used only mappings present in the local cache will be included in the results given. `crds.uses` is invoked as:

```
% python -m crds.uses <observatory=hst|jwst> <mapping or reference>...
```

e.g.:

```
% python -m crds.uses hst s7g1700gl_dead.fits
hst.pmap
hst_cos.imap
hst_cos_deadtab.rmap
```

5.5 `crds.matches`

`crds.matches` reports the match patterns which are associated with the given reference files:

Usage: matches.py [options] <context> <references...>

Options:

-h, --help	show this help message and exit
-f, --full	Show the complete match path through the mapping hierarchy.
-V VERBOSITY, --verbose=VERBOSITY	Set verbosity level.

crds.matches can be invoked as:

```
% python -m crds.matches hst.pmap o8u2214fj_dxy.fits
('HRC', 'CLEAR1S', 'F220W')

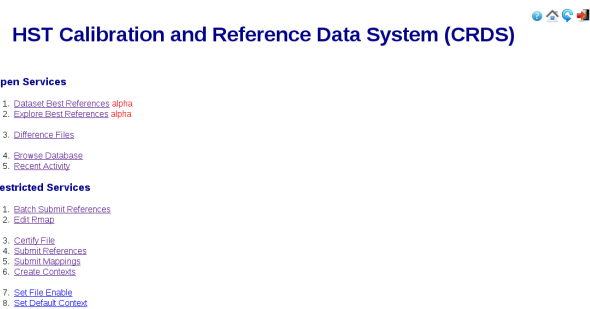
% python -m crds.matches --full hst.pmap o8u2214fj_dxy.fits
('hst', 'acs', 'dgeofile', 'HRC', 'CLEAR1S', 'F220W', '2002-03-01', '00:00:00')
```

5.6 crds.sync

crds.sync downloads references and mappings from the CRDS server based on a variety of specification mechanisms.

USING THE CRDS WEB SITE

CRDS has websites at hst-crds.stsci.edu and jwst-crds.stsci.edu which support the submission, use, and distribution of CRDS reference and mappings files. Functions on the CRDS website are either public functions which do not require authentication or private functions which require a CRDS login account.



Functions annotated with the word (alpha) are partially completed components of a future build which may prove useful now.

6.1 Public Functions

The following functions are available for anyone with access to the CRDS web server and basically serve to distribute information about CRDS files and recommendations.

6.1.1 Dataset Best References (alpha)

The *Dataset Best References* page supports determining the best references for a single dataset with respect to one CRDS context. Best references are based upon a CRDS context and the parameters of the dataset as determined by the dataset file itself or a database catalog entry.

Context

The context defines the versions set of CRDS rules used to select best references. *Edit* is the default context from which most newly created contexts are derived. *Operational* is the context currently in use by the pipeline. *Recent* shows the most recent contexts. *User Specified* enables the submitter to type in the name of any other known context.

HST Calibration and Reference Data System (CRDS)

Dataset Best References

Context:

Editing or more recent

@ Operational or more recent

@ Recent

@ User Specified * 0 het_0001.pmap

Dataset:

Upload FITS Header No file chosen

@ Upload Dataset No file chosen

@ Archived Dataset ID e.g. R0700010

Dataset Best References lets you determine the best reference files for an archived or uploaded dataset, relative to the chosen CRDS context.
Context this section defines the version of the pipeline context that should be used to determine the best reference files. The context file chosen defines the parameter choices which will be available as well as complete file selection criteria.
Dataset Dataset specifies the name of an archived or uploaded dataset for which the user needs to know the best reference files that are defined by the chosen context. Upload FITS header uses your browser to minimize file information uploaded to the server for FITS files.

Dataset

Upload FITS header

Browser-side code can extract the FITS header of a dataset and upload it to the server where best references are computed based on dataset parameters. This function is implemented in Javascript and reliant on HTML5; it supports only parameters present in the FITS primary header. It avoids uploading most of the dataset. It is known to work in Firefox and Chrome but not IE or Safari-5.

Upload Dataset

A user's dataset can be uploaded to the server for best references evaluation.

Archived Dataset

Datasets can be specified by ID and their best reference input parameters will be retrieved from the catalog.

Dataset Best References Results

HST Calibration and Reference Data System (CRDS)

Best References iaai01rtq_raw.fits

Input Parameters		Recommended Files	
APERTURE:	UVIS	ATODTAB:	q9016386_atd.fits download
BNAXIS1:	1.0	BIASFILE:	u1r1346n_bia.fits download
BNAXIS2:	1.0	BPXTAB:	v5420120_bpw.fits download
CCDAMP:	ABCD	CCDTAB:	t281658rm_ccd.fits download
CCDSAIN:	1.5	CRREJTAB:	q9014301_crr.fits download
CHINJECT:	NONE	DARKFILE:	t34201771_drk.fits download
DATE-OBS:	2009-07-14	FLSHFILE:	w701705d_fls.fits download
DETECTOR:	UVIS	IDCTAB:	w201956n_idc.fits download
FILTER:	F555W	MDRIZTAB:	ub1853g_mdr.fits download
INSTRUME:	WFC3	NULFILE:	n/a
REFTYPE:	UNDEFINED	OSCENTAB:	q911321ol_osc.fits download
SAMP_SEQ:	UNDEFINED	PFLTFILE:	v5816168_pfl.fits download
SUBARRAY:	F		
SUBTYPE:	UNDEFINED		
TIME-OBS:	15:56:09	Download All:	iaai01rtq_raw_bestrefs.tar.gz

The results page for dataset best references displays the input parameters which were extracted from the dataset header on the right side of the page.

Best reference recommendations are displayed on the left side of the page.

6.1.2 Explore Best References (alpha)

Explore Best References supports entering best references parameters directly rather than extracting them from a dataset or catalog. Explore best references is essentially a sand box which lets someone evaluate what CRDS will do

given particular parameter values. The explorer currently lists all parameters which might be relevant to any mode of an instrument and has no knowledge of default values.

The first phase of exploration is to choose a pipeline context and instrument which will be used to define parameter choices:

The second phase is to enter the parameters of a dataset which are relevant to best references selection.

The entered parameters are evaluated with respect to the given pipeline context and best references are determined. The results are similar or identical to the *Dataset Best References* results.

6.1.3 Difference Files

Difference Files can be used to compare two reference or mapping files. Either the name of a file already in CRDS can be specified (known) or any file can be uploaded via the web (uploaded).

For mappings, *Difference Files* displays two kinds of information:

- logical differences where CRDS analyzes the mappings and reports the parameter trail and effect of the difference (add, delete, replace).

HST Calibration and Reference Data System (CRDS)

Difference hst.pmap against hst_0019.pmap

Mapping Logical Differences

- Logical

Mapping Text Differences

- (hst.pmap, hst_0019.pmap)
- (hst_acs.imap, hst_acs_0016.imap)
- (hst_acs_darkfile.imap, hst_acs_darkfile_0011.imap)
- (hst_cos.imap, hst_cos_0002.imap)
- (hst_cos_deadtab.imap, hst_cos_deadtab_0003.imap)
- (hst_wfc3.imap, hst_wfc3_0001.imap)
- (hst_wfc3_biasfile.imap, hst_wfc3_biasfile_0001.imap)

HST Calibration and Reference Data System (CRDS)

Difference hst.pmap against hst_0019.pmap

Mapping Logical Differences

Logical					
(hst.pmap, hst_0019.pmap)	(hst_acs.imap, hst_acs_0016.imap)	(hst_acs_darkfile.imap, hst_acs_darkfile_0011.imap)	(HRC; XJABCDJ40RBJCJDP; 1.012-0.94-0.01.0)	2002-03-18 00:00:00	replaced hst_00202q_dsk.fits with hst_acs_darkfile_0012.fits
(hst.pmap, hst_0019.pmap)	(hst_acs.imap, hst_acs_0016.imap)	(hst_acs_dgnofile.imap, hst_acs_dgnofile_0005.imap)	(HRC; CLEARIS; F220W)	2002-03-01 00:00:00	replaced hst_02214g_dsk.fits with hst_acs_dgnofile_0002.fits
(hst.pmap, hst_0019.pmap)	(hst_acs.imap, hst_acs_0016.imap)	(hst_acs_dgnofile.imap, hst_acs_dgnofile_0005.imap)	(HRC; F475W; CLEARIS)	2002-03-01 00:00:00	replaced hst_02214g_dsk.fits with hst_acs_dgnofile_0005.fits
(hst.pmap, hst_0019.pmap)	(hst_cos.imap, hst_cos_0002.imap)	(hst_cos_deadtab.imap, hst_cos_deadtab_0003.imap)	(FUV)	1996-10-01 00:00:00	replaced hst_017img_dead.fits with hst_cos_deadtab_0003.fits
(hst.pmap, hst_0019.pmap)	(hst_cos.imap, hst_cos_0002.imap)	(hst_cos_deadtab.imap, hst_cos_deadtab_0003.imap)	(FUV)	1997-10-01 01:01:01	added hst_cos_deadtab_0002.fits
(hst.pmap, hst_0019.pmap)	(hst_wfc3.imap, hst_wfc3_0001.imap)	(hst_wfc3_biasfile.imap, hst_wfc3_biasfile_0001.imap)	(UNUS; NH[CID; '1.5', '1.0', '1.0', UNUS-QUAD-SUBJ[AVS1-2K2A-SUBJ[AVS1-2K2B-SUBJ[AVS1-C3L2A-SUBJ[AVS1-C3L2B-SUBJ[AVS2-2K2C-SUBJ[AVS2-2K2D-SUBJ[AVS2-C3K1C-SUBJ[AVS2-C512C-SUBJ[AVS2-C512D-SUBJ[AVS2-M3K1C-SUBJ[AVS2-M512C-SUBJ; NONE; NaN)	2010-12-10 00:00:00	replaced v517169_bias.fits with hst_wfc3_biasfile_0001.fits

- textual differences which show the context difference (diff -c) of the two mapping files.

Mapping Text Differences

```

- (hst.pmap, hst_0019.pmap)

--- hst.pmap      2012-09-08 08:36:20.352312846 -0500
+++ hst_0019.pmap 2012-09-14 14:15:59.820502952 -0500

@@ -1,18 +1,18 @@
header = {
-  'name' : 'hst.pmap',
+  'derived_from' : 'created by hand 12-23-2011',
+  'name' : 'hst_0019.pmap',
+  'derived_from' : 'hst_0018.pmap',
+  'mapping' : 'PIPELINE',
+  'observatory' : 'HST',
+  'parkey' : ('INSTRUME',),
+  'description' : 'Initially generated on 2011-11-16 10:29:00',
+  'sha1sum' : 'e2c0392fd2731df1e8d932bd996f7fd319a813db',
+  'sha1sum' : '72d9b3025e2e884896aa7a5e3e5d25de17f18820',
+ }

selector = {
-  'ACS' : 'hst_acs.imap',
+  'COS' : 'hst_cos.imap',
+  'ACS' : 'hst_acs_0016.imap',

```

6.1.4 Browse Database

The *Browse Database* feature enables examining the metadata and computable properties of CRDS reference and mapping files.

The first phase is to enter a number of filters to narrow the number or variety of files which are displayed. Leaving any filter at the default value of * renders that constraint irrelevant and all possible files are displayed with respect to that constraint. The result of the first phase is a table of files which matched the filters showing their basic properties.

The second phase is initiated by clicking on the filename link of any file displayed in the table from the first phase. Clicking on a filename link switches to a detailed view of that file only:

HST Calibration and Reference Data System (CRDS)

Browse Database

Filters

Instrument:

Reference Type:

Filename:

User:

Status:

Extension:

Browse Database Searches the CRDS database for files matching the criteria you specify. Filters restrict the search to files matching that value. To ignore a field during search, set it to *.

HST Calibration and Reference Data System (CRDS)

Browse Database

Filters

filter	value
instrument	acs

Database Entries

submit date	name	status	submitter	description	instrument	filekind
2012-09-08 08:39	hst_acs_imap	delivered	jmler	Initial import	acs	
2012-09-08 08:39	hst_acs_atodtab.rmap	delivered	jmler	Initial import	acs	atodtab
2012-09-08 08:39	hst_acs_biastab.rmap	delivered	jmler	Initial import	acs	biastab
2012-09-08 08:39	hst_acs_bpictab.rmap	delivered	jmler	Initial import	acs	bpictab
2012-09-08 08:39	hst_acs_ccdtab.rmap	delivered	jmler	Initial import	acs	ccdtab
2012-09-08 08:39	hst_acs_ctfile.rmap	delivered	jmler	Initial import	acs	ctfile
2012-09-08 08:39	hst_acs_crestab.rmap	delivered	jmler	Initial import	acs	crestab
2012-09-08 08:39	hst_acs_d2mfile.rmap	delivered	jmler	Initial import	acs	d2mfile
2012-09-08 08:39	hst_acs_darkfile.rmap	delivered	jmler	Initial import	acs	darkfile
2012-09-08 08:39	hst_acs_dgeofile.rmap	delivered	jmler	Initial import	acs	dgeofile
2012-09-08 08:39	hst_acs_dkctfile.rmap	delivered	jmler	Initial import	acs	dkctfile
2012-09-08 08:39	hst_acs_fishfile.rmap	delivered	jmler	Initial import	acs	fishfile
2012-09-08 08:39	hst_acs_idctab.rmap	delivered	jmler	Initial import	acs	idctab
2012-09-08 08:39	hst_acs_imphtab.rmap	delivered	jmler	Initial import	acs	imphtab
2012-09-08 08:39	hst_acs_mdctab.rmap	delivered	jmler	Initial import	acs	mdctab

HST Calibration and Reference Data System (CRDS)

Browse [hst_acs_darkfile.rmap](#) [edit](#) [download](#)

Database
Contents
Past Actions
Used By Files

The database details page has a number of accordion panes which open when you click on them. All file types have these generic panes:


- Database - lists a table of CRDS metadata for the file.
- Contents - shows the text of a mapping or internal details about a reference file.
- Past Actions - lists website actions which affected this file.
- Used By Files - list files which reference this file.

Reference files have these additional panes:

- Certify Results - shows the results of `crds.certify` run on this reference now.
- Lookup Patterns - lists the parameters sets which lead to this reference.

6.1.5 Recent Activity

The *Recent Activity* view shows a table of the actions on CRDS files which are tracked. Only actions which change the states of files in some way are tracked:

HST Calibration and Reference Data System (CRDS) 

List Actions Recent Activity Search the CRDS database for actions matching the filters you specify. To ignore a field during search, set it to *.

Filters

Action:

Instrument:

Reference Type:


Filename:

User:

Status:

Extension:

The first page lists a number of constraints which can be used to choose activities of interest. To ignore any constraint, leave it set at the default value of *. The result of the activity search is a table of matching actions:

HST Calibration and Reference Data System (CRDS) 

Recent Activity

Filters

filter	value
action	submit file
instrument	acs

Activity

date	filename	action	user	why	details
2012-09-10 20:05	hst_acs_dadfile_0007.fits	submit file	pmiller	Something.	{{{"v4414346k_drk.fits", "hst_acs_dadfile_0007.fits"}, "ID"}}

The details vary by the type of action, in this case showing the original name of a file prior to submission to CRDS and the assignment of its official name.

6.2 Private Functions

The following functions are restricted to users with accounts on the CRDS website and support the submission of new reference and mapping files and maintenance of the overall site.

6.2.1 Batch Submit References

Batch Submit References is intended to handle the majority of CRDS reference submissions with a high degree of automation. This page accepts a number of reference files and metadata which is applied to all of them. The specified reference files are checked on the server using `crds.certify` and if they pass are submitted to CRDS. All of the submitted references must be of the same reference type, i.e. controlled by the same `.rmap` file.

HST Calibration and Reference Data System (CRDS)

Batch Submit Reference

Old Context to Derive From

#Editing

hst_0019.pmap 2012-09-14 14:15

or more recent

Operational

hst.pmap 2012-09-08 09:39

or more recent

@Recent

hst.pmap 2012-09-08 09:39

or more recent

@User Specified

4.8 hst_0001.pmap

Change Level:

SEVERE

Creator:

Description

Submitted References

#Uploaded Files

Choose File: No file chosen

@Server Directory

Auto-Rename:

Submit References

Batch Submit References supports uploading a list of reference files for a single instrument and reference type, i.e. one .pmap. After checking and storing the reference files, new CRDS mapping files which refer to them are generated.

Old Context to Derive From The name of the pipeline context which will be recursively updated to insert the new reference files.

Change Level The degree to which the new files are expected to impact science results.

Description Information about what's new in this file and what the expected impacts are.

Creator author of the contents or changes to the file, not necessarily the submitter. The submitter is known based on login information.

Submitted References Uploaded Files lets you upload a list of new references to CRDS. Each reference file must certify on the entire batch will be rejected. If you are a member of group ctdsoper, you can make your own subdirectory in /gpcrdshstref and copy your submitted files there, then refresh this page and select Server Directory and the directory you created from the drop-down menu.

Auto Rename when checked, the uploaded file will be renamed to a unique name using CRDS naming conventions and id numbers, e.g. s7g1700gm_deadtab might be renamed as hst_cos_deadtab_0002.fits. This option enables easy comparisons with CRDS names during side-by-side testing with CRDS. Once CRDS is operational Auto Rename should be left unchecked so that CRDS-style names are assigned.

The specified context is used as the starting point for new automatically generated context files and also determines any predecessors of the submitted references for comparison during certification. If all the submitted reference files pass certification, a new `.rmap`, `.imap`, and `.pmap` are all generated automatically to refer to the newly entered references. Based on their header parameters, references are automatically assigned to appropriate match locations in the `.rmap` file.

HST Calibration and Reference Data System (CRDS)

Batch Reference Submit Results

Starting Context

hst_0021.pmap

Certify Results

s7g1700gm_dead.fits --> hst_cos_deadtab_0006.fits OK

Generated New Mappings

hst_cos_deadtab_0006.rmap

hst_0022.pmap

hst_cos_0005.imap

Actions on hst_cos_deadtab_0005.rmap

Action	New Reference File	File Match	At	Rmap Match	User after	Replaced File
REPLACE	hst_cos_deadtab_0006.fits	(FUV)	-	(FUV)	1997-10-01 01:01:01	hst_cos_deadtab_0002.fits

Rmap Differences

Confirm or Abort Submission

confirm

discard

Reference Certification

In this case the user submitted a single COS deadtab file named `s7g1700gm_dead.fits` which was added to context `hst_0021.pmap`. As indicated in the Certify Results accordion panel, `s7g1700gm_dead.fits` was renamed to `hst_cos_deadtab_0006.fits` in CRDS and certified OK. Opening the accordion panel displays the results of the `crds.certify` including provenance information. Changes in table mode coverage will show up as Certify Results warnings.

Any certification error will result in the failure of the entire batch submission and will redirect back to the input page with a single error message.

Batch submitting `s7g1700gl_dead.fits` added it to CRDS and generated three new mapping files which were derived from `hst_0019.pmap`: `hst_cos_deadtab_0006.rmap`, `hst_cos_0005.imap`, `hst_0022.pmap`.

Rmap Updates

The `rmap` under `hst_0021.pmap` that corresponds to `s7g1700gm_dead.fits` was `hst_cos_deadtab_0005.rmap`. To add `s7g1700gm_dead.fits`, its header parameters were matched against `hst_cos_deadtab_0005.rmap` to determine where it should be added. *Actions on hst_cos_deadtab_0005.rmap* shows that the new reference file replaced `hst_cos_deadtab_0002.fits`. The *Rmap Differences* accordion shows the textual differences between `hst_cos_deadtab_0005.rmap` and `0006`.

Collisions

Under some circumstances, a *Collision Warning* accordion will be present. It should be carefully examined to ensure that overlapping edits of the same context file have not occurred. Overlaps can be resolved by cancelling the current submission and re-doing it, or by accepting the current submission and manually correcting the mappings involved. Failure to correctly resolve a collision will most likely result in one of two sets of conflicting changes being lost.

Confirm or Discard

If everything looks good the last step is to click the *Confirm* button. Clicking the Confirm button finalizes the submission process, submits the files for archive pickup, and makes them a permanent part of CRDS visible in the database browser and potentially redistributable. A confirmed submission cannot be revoked, but neither will it go into use until the pipeline or a user explicitly requests it.

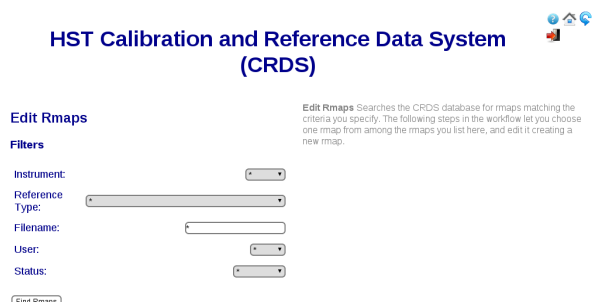
Discarding a batch submission based on warnings or bad `rmap` modifications removes the submission from CRDS. In particular temporary database records and file copies are removed.

If a submission is neither confirmed nor cancelled the files involved will be automatically removed, nominally after two weeks.

Following any CRDS pipeline mapping submission, the default *edit* context is updated to that pipeline mapping making it the default starting point for future submissions.

6.2.2 Edit Rmap

Edit Rmap provides a Javascript based `rmap` editor which supports adding or replacing new `rmap` rows. The existing prototype can add new `USEAFTER` rows but cannot add new `Match` tuple cases. The prototype works only on HST-style `rmaps` which consist of an outer `Match()` selector and a number of inner `UseAfter()` selectors.



The screenshot shows the 'Edit Rmaps' interface of the HST Calibration and Reference Data System (CRDS). The title 'HST Calibration and Reference Data System (CRDS)' is at the top in blue, with a small icon to its right. Below the title, the 'Edit Rmaps' section is visible. It includes a 'Filters' section with several dropdown menus: 'Instrument:', 'Reference Type:', 'Filename:', 'User:', and 'Status:'. A 'Find Rmaps' button is located at the bottom of the filters. To the right of the filters, there is a text box containing the following text: 'Edit Rmaps Searches the CRDS database for maps matching the criteria you specify. The following steps in the workflow let you choose one rmap from among the rmaps you list here, and edit it creating a new rmap.'

The first step is to select an rmap to edit by choosing filters of rmaps to list and then clicking on one of the links in the *name* column:

HST Calibration and Reference Data System (CRDS)

Choose an Rmap to Edit

Filters

filter	value
instrument	acs
extension	rmap

Rmap Choices

delivery date	name	status	submitter	description	instrument	filekind
2012-09-08 08:39	hst_acs_atodtab.rmap	delivered	jmiller	Initial import	acs	atodtab
2012-09-08 08:39	hst_acs_biasfile.rmap	delivered	jmiller	Initial import	acs	biasfile
2012-09-08 08:39	hst_acs_bptab.rmap	delivered	jmiller	Initial import	acs	bptab
2012-09-08 08:39	hst_acs_ccdtab.rmap	delivered	jmiller	Initial import	acs	ccdtab
2012-09-08 08:39	hst_acs_crtfile.rmap	delivered	jmiller	Initial import	acs	crtfile
2012-09-08 08:39	hst_acs_crrtab.rmap	delivered	jmiller	Initial import	acs	crrtab
2012-09-08 08:39	hst_acs_d2imfile.rmap	delivered	jmiller	Initial import	acs	d2imfile
2012-09-08 08:39	hst_acs_darkfile.rmap	delivered	jmiller	Initial import	acs	darkfile
2012-09-08 08:39	hst_acs_dgeofile.rmap	delivered	jmiller	Initial import	acs	dgeofile

The second step is to add or replace file lines in an rmap by clicking on them:

```
selector = Match([
  ('HRC', 'A|BCD|AD|B|BC|C|D', '1.0|2.0|4.0|8.0') : UseAfter([
    '1992-01-01 00:00:00' : 'lcb12060j_drk.fits',
    '2002-03-01 00:00:00' : 'n3o1022cj_drk.fits',
  ])
])
```

+ replace

Clicking on a line exposes two buttons adjacent to it: + and *replace*.

Edit Rmap

Click on a useafter date to edit the rmap at that location.

```
header = {
  'derived_from' : 'generated from CRDS database 2012-08-10 15:26:07.331701',
  'filekind' : 'DARKFILE',
  'instrument' : 'ACS',
  'mapping' : 'REFERENCE',
  'name' : 'hst_acs_darkfile.rmap',
  'observatory' : 'HST',
  'parkey' : ['(DETECTOR, 'CCDAMP', 'CCDGA1W'), ('DATE-OBS', 'TIME-OBS')],
  'parkey_relevance' : {
    'ccdamp' : '(DETECTOR != "SBC")',
    'ccdgain' : '(DETECTOR != "SBC")',
  },
  'rmap_relevance' : 'ALWAYS',
  'sha1sum' : 'c0ee170905f8050bc89dc878afc17ef4fd73115',
}
```

```
selector = Match([
  ('HRC', 'A|BCD|AD|B|BC|C|D', '1.0|2.0|4.0|8.0') : UseAfter([
    '1992-01-01 00:00:00' : 'lcb12060j_drk.fits',
    '2002-03-01 00:00:00' : 'n3o1022cj_drk.fits',
    '2002-03-18 00:00:00' : 'n3o1022cj_drk.fits',
    '2002-03-19 00:34:31' : 'n3o1022cj_drk.fits',
    '2002-03-20 00:34:32' : 'n3o1022cj_drk.fits',
    '2002-03-21 00:34:31' : 'n3o1022cj_drk.fits',
    '2002-03-22 00:34:30' : 'n3o1022cj_drk.fits',
    '2002-03-23 00:34:30' : 'n3o1022cj_drk.fits',
  ])
])
```

+ replace

Choose File | No file chosen Change Level: SEVERE + delete

New lines are added by clicking on the +. This will open a line with a text entry box for entering a USEAFTER date and a file input as well as some additional parameters. A newly submitted reference file can be entered into the file input box and will be uploaded and submitted to CRDS when the page is submitted.

Clicking *replace* strikes out the clicked line and adds inputs below it at the same USEAFTER time.

Additional lines can be added by clicking the + adjacent to an added line. An added line can be removed by clicking the *delete* button adjacent to it. A replaced line can be restored by clicking on *unreplace*.

When the edited rmap is submitted, the supplied reference files are uploaded and certified.

6.2.3 Certify File

Certify File runs `crds.certify` on a file currently in CRDS (known) or on a file uploaded to the server.

If the certified file is a reference table, the specified context is used to locate a comparison file. *Certify File* can be used to check files already in CRDS.

HST Calibration and Reference Data System (CRDS)

Certify File

Context to Check Against

* Editing: hst_0023.pmap 2012-09-14 16:15
 or more recent
 @ Operational: hst.pmap 2012-09-08 08:39
 or more recent
 @ Recent: **hst.pmap 2012-09-08 08:39**
 @ User Specified:

Checked File

* Uploaded File: Choose File | No file chosen
 @ Known File:

Certify File runs the same checks required for submission and presents any errors as well as provenance information for review.

Context to Check Against defines any prior reference used to check modes of a new reference.

Uploaded File lets you temporarily upload a prototype reference or mapping to the CRDS server for certification.

Known File lets you re-certify a mapping or reference which has already been submitted to CRDS.

NOTE: if you have installed CRDS, you can certify files locally like this:

```
% python -m crds.certify ./hst_acc_biasfile_0001.rmap
% python -m crds.certify --context=hst.pmap --dump-provenance ./myfile.fits
```

WARNING: certification of context files, pmaps and imaps, is a transitive check so it may certify many mappings and can be slow.

6.2.4 Submit References

Submit References provides a basic interface for submitting a list of references which don't have to be related. No context mappings are generated to refer to the submitted files. Submitted references must still pass through `crds.certify`.

HST Calibration and Reference Data System (CRDS)

Submit Reference File

Context to Check Against

* Editing: hst_0023.pmap 2012-09-14 16:15
 or more recent
 @ Operational: hst.pmap 2012-09-08 08:39
 or more recent
 @ Recent: **hst.pmap 2012-09-08 08:39**
 @ User Specified:

Submitted Files:

* Uploaded Files: Choose File | No file chosen
 @ Server Directory:

Change Level: **SEVERE**Creator:

Description:

Auto-Rename: ☐

Submit Files permanently enters files into CRDS and the archive.

Context to Check Against defines any prior reference used to check modes of a new reference.

Submitted Files new references or mappings to be added to CRDS.

Change Level The degree to which the new files are expected to impact science results.

Description information about what's new in this file and what the expected impacts are.

Creator author of the contents or changes to the file, not necessarily the submitter. The submitter is known based on login information.

Auto Rename when checked, the uploaded file will be renamed to a unique name using CRDS naming conventions and id numbers, e.g. 'STp1700gm_deadtab_0002.fits'. This option enables easy comparisons with CBES names during side-by-side testing with CRDS. Once CRDS is operational Auto Rename should be left unchecked so that CRDS-style names are assigned.

NOTE: All submitted files must pass certification before they will be accepted by CRDS. All references referred to by a mapping must be submitted before the mapping will be accepted.

6.2.5 Submit Mappings

Submit Mappings provides a basic interface for submitting a list of mapping files which don't have to be related. This can be used to submit context files which refer to files from *Submit References* and with fewer restrictions on allowable changes. Typically only .rmaps are submitted this way.

6.2.6 Create Contexts

Create Contexts provides a basic interface for automatically generating pipeline and instrument context mappings which refer to the specified reference mapping files.

Using *Create Contexts* the upper level mappings can be modified to refer to a number of (most likely hand-edited) reference mappings. Rmaps referred to by create contexts must already be known to CRDS and can be typed into the

HST Calibration and Reference Data System (CRDS)

Submit Mapping File

Submitted Files:

☒ Uploaded Files
☐ Server Directory

Change Level:

Creator:

Description:

Auto-Rename: ☐

Submit File

Submit Files permanently enters files into CRDS and the archive.

Submitted Files new references or mappings to be added to CRDS.

Change Level The degree to which the new files are expected to impact science results.

Description Information about what's new in this file and what the expected impacts are.

Creator author of the contents or changes to the file, not necessarily the submitter. The submitter is known based on login information.

Auto Rename when checked, the uploaded file will be renamed to a unique name using CRDS naming conventions and id numbers, e.g. '97p1700gm_dead.fir' might be renamed as 'hst_cos_deadtab_0002.fir'. This option enables easy comparisons with CDSS names during side-by-side testing with CRDS. Once CRDS is operational Auto Rename should be left unchecked so that CRDS-style names are assigned.

NOTE: All submitted files must pass certification before they will be accepted by CRDS. All references referred to by a mapping must be submitted before the mapping will be accepted.

HST Calibration and Reference Data System (CRDS)

Create Parent Contexts

Derived From

☒ Editing hst_0023.pmap 2012-09-14 16:15
☐ Operational hst.pmap 2012-09-08 08:39
☐ Recent
☐ User Specified

New Rmaps

Description

Create Parent Contexts

Create Parent Contexts automatically generates and submits new pipeline and instrument contexts which refer to your new rmaps.

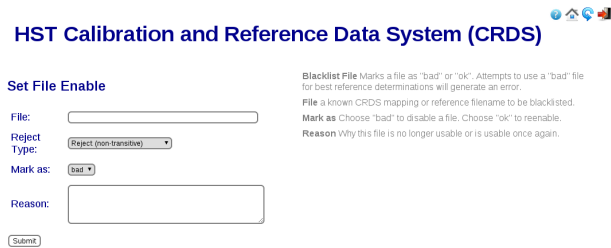
Derived From defines the context which will be modified to create a new context. The pipeline context refers to a set of instrument contexts, new instrument contexts will be derived from the existing contexts by updating them to refer to the new Rmaps. A new pipeline context will be generated to refer to the new instrument contexts.

New Rmaps will replace their equivalents in the instrument contexts identified by the pipeline context. Each file in New Rmaps must have already been accepted by CRDS. New Rmaps can be listed one file per line, separated by spaces, or separated by commas.

Description should document the general rationale or purpose behind the set of New Rmaps.

6.2.7 Set File Enable

Set File Enable provides control over the Blacklist and Reject attributes of a file.



The screenshot shows the 'Set File Enable' form in the HST Calibration and Reference Data System (CRDS). The form is titled 'Set File Enable' and includes the following fields and controls:

- File:** A text input field.
- Reject Type:** A dropdown menu with 'Reject (non-transitive)' selected.
- Mark as:** A dropdown menu with 'bad' selected.
- Reason:** A text input field.
- Submit:** A button at the bottom left.

On the right side of the form, there are four informational text blocks:

- Blacklist File:** Marks a file as "bad" or "ok". Attempts to use a "bad" file for best reference determinations will generate an error.
- File:** A known CRDS mapping or reference filename to be blacklisted.
- Mark as:** Choose "bad" to disable a file. Choose "ok" to reenable.
- Reason:** Why this file is no longer usable or is usable once again.

Rejecting a file is used to signal that the file should no longer be used. Rejecting a file affects only that file. Blacklisting a file marks the file as unusable, but it also blacklists all files which directly or indirectly refer to the original blacklisted file. So, blacklisting is transitive, but rejection is intransitive. Either blacklisting or rejection can be undone by marking the file as OK again using *Set File Enable*. Only files which are already known to CRDS can be rejected or blacklisted.