Standards Change Request

File Checksums **SCR3-1034.v9**
Elizabeth D. Rye **July 12, 2006**

Provenance:

Date: 2006-07-11, revision 8.0
Working Group: E. Rye
Title: File Checksums (SCR3-1034.v8)

Date: 2006-05-16, revision 7.0
Working Group: E. Rye, R. Simpson
Title: File Checksums (SCR3-1034.v7)

Date: 2006-05-15, revision 6.0
Working Group: E. Rye
Title: File Checksums (SCR3-1034.v6)

Date: 2006-03-22, revision 5.0
Working Group: E. Rye (lead), T. King, M. McAuley
Title: File Checksums (SCR3-1034.v5)

Date: 2006-03-20, revision 4.0
Working Group: E. Rye (lead), T. King, M. McAuley
Title: File Checksums (SCR3-1034.v4)

Date: 2006-03-06, revision 3.0
Working Group: E. Rye (lead), T. King, M. McAuley
Title: File Checksums (SCR3-1034.v3)

Date: 2005-11-14, revision 2.0
Working Group: T. King (lead), M. McAuley
Title: MD5 Checksums (SCR3-1034.v2)

Date: 2004-11-22, revision 1.0
Working Group: J. Wilf (lead), T. King, M. McAuley
Title: MD5 Checksums for Files (SCR3-1034.v1)

Problem:

As an entity responsible for maintaining data, the PDS must be able to ascertain the integrity of its archive. The following requirements are part of the PDS Requirements document which was approved by the Management Council on May 26, 2006:

1.3.3 PDS will provide criteria for validating archival products
2.7.3 PDS will provide standard protocols for accessing data, metadata and computing resources across the distributed archive
2.8.3 PDS will meet U.S. federal regulations for the preservation and management of data.
3.2.3 PDS will provide mechanisms to ensure that data have been transferred intact (Note that 3.2.1 and 3.2.2 specify that mechanisms must be available to transfer data to users both online and offline.)
4.1.2 PDS will develop and implement procedures for periodically ensuring the integrity of the data.
4.1.3 PDS will develop and implement procedures for periodically refreshing the data by updating the underlying storage technology
4.1.5 PDS will meet U.S. federal regulations for preservation and management of the data through its Memorandum of Understanding (MOU) with the National Space Science Data Center (NSSDC)

From the above we may derive the following specific responsibilities:

1. verifying the integrity of data stored on physical media (4.1.2),
2. detecting errors introduced during transfer of data to newer media (4.1.3),
3. detecting errors that occur during the transmission of data from data providers to the PDS (1.3.3), between PDS nodes (2.7.3), from the PDS to the NSSDC (2.8.3 and 4.1.5), and from the PDS to end users (3.2.3).

At present, there are no generally accepted methods within the PDS for meeting any of these responsibilities. The purpose of this SCR is to address the problem of how to *detect* errors that have been introduced into data. The purpose of this SCR is not to address the problem of how to *restore* data that have become corrupted. (That problem is addressed by maintaining multiple copies and appropriate backups of data.)

Proposed Solution:

Require a list of MD5 checksums on every PDS archive volume

A simple method for detecting errors is to create and maintain a list of checksum values for every file contained in a PDS archive volume. The checksum values can be confirmed periodically for static PDS holdings or at the completion of a data transfer.

This SCR outlines a procedure whereby such checksum lists will be included in a prescribed manner within a PDS archive volume. An "archive volume" for these purposes is taken to be a collection of information that:

- conforms to the structure defined in Chapter 19 of the PDS Standards Reference
- may be either physical or logical
- has successfully passed peer review
- is ready for permanent placement in the archive
- will subsequently be sent to the NSSDC for deep archive
- upon changing, will require a new version number for at least the volume, if not the data set.

The proposal outlined here, while permitting only the MD5 checksum type, may be amended in the future based on needs and experience.

The changes proposed in this SCR are to:

1. Establish a required, reserved file, "CHECKSUM.TAB", which contains MD5 checksum values for all files in an archive volume, to be included in the INDEX directory.
2. Create a new keyword, CHECKSUM_TYPE, for use in the CHECKSUM.LBL file, to specify the type of checksum algorithm used to generate the values in the checksum file.
3. Update the element definition for the MD5_CHECKSUM keyword, changing the STATUS_TYPE to APPROVED.

Requested Changes:

<u>Changes to the Standards Reference</u>

The following changes to the PDS Standards Reference are required to support this SCR:

Add to section 10.2.2 Reserved File Names, "CHECKSUM.TAB".

Add to Chapter 19.3.2.3 INDEX Subdirectory, after INDEX.TAB:

CHECKSUM.LBL **Required**

This is the PDS label for the CHECKSUM.TAB file. The object definition for the CHECKSUM column must contain the CHECKSUM_TYPE keyword.

CHECKSUM.TAB **Required**

This file contains an MD5 checksum for every file on the volume except itself and its label. The file is a PDS ASCII TABLE object with two columns, one (named CHECKSUM) providing the checksum values and another (named FILE_SPECIFICATION_NAME) containing the path and name of each file in the archive relative to the root directory of the volume.

For examples of the CHECKSUM.TAB and CHECKSUM.LBL files, see Appendix D, section D.2.

Each figure in Chapter 19, Volume Organization and Naming, will need to be updated to include “CHECKSUM.TAB” and "CHECKSUM.LBL" files in the INDEX directory.

Appendix D, section D.2 add the sample CHECKSUM.TAB and CHECKSUM.LBL files as shown in the attachment. (The following sections of Appendix D will need to be re-numbered.)

Changes to the Data Dictionary

Modify the description of the MD5_CHECKSUM keyword as shown in the attached element definition template.

Add the new keyword, CHECKSUM_TYPE, as shown in the attached element definition template.

Changes to the PDS Tool Suite

A number of utilities are already widely available which can be used to produce and read the CHECKSUM.TAB file. However, any tool which validates PDS volumes (*e.g.*, the volume verifier) will need to be modified to determine whether CHECKSUM.TAB is present and to carry out the checksum verifications.

Impact Assessment:

The impact of this SCR will vary depending on how individual nodes choose to generate the checksum lists.

Some PDS nodes may require the generation of these files by their data providers, in which case the data providers will have the responsibility to implement a checksum generation step somewhere in their ground data processing stream. (This should be a negligible impact to the missions if this expectation is clearly understood from the beginning of the software development phase.)

Other nodes may choose not to levy any requirements on their data providers and choose to generate the checksum files themselves. In its simplest implementation, this could involve running an off-the-shelf utility like md5deep against individual data volumes and generating a PDS label for the resulting file. A more sophisticated approach might involve wrapping a utility like md5deep in a simple script that would traverse one or more data volume directory trees and generate the appropriate checksum files and their labels. Such a script could also be modified to append to existing checksum files if data are added to existing non-archive volumes. A rough estimate of the time required to generate such a script is approximately eight hours of labor per node.

If nodes should choose to implement checksums as a method for validating the integrity of their entire data holdings, they will need, at some point, to go back and generate checksum values for all of their archive volumes that were created without checksums. Assuming no pre-existing script, a rough estimate of the time to accomplish this task is again about eight hours of labor per node. Further effort (perhaps four hours) would be required to generate a script to validate the integrity of existing archive holdings periodically using the checksum values previously generated. (No implication is intended that these older data holdings need to be re-mastered to include the checksum lists within the archive volumes. It should be entirely adequate to maintain these lists separately until such time as the data are transferred to newer media for media refreshment purposes.)

At some point in the future (again, not mandated by this SCR), a mechanism associated with the product servers may be generated on a best efforts basis to extract checksum values from checksum files and provide them to users who download individual PDS data product files.

Discussion:

Does the proposed solution address the requirements for data integrity checking by the PDS specified in the "Problem" section?

Here, once again, is a list of the areas of PDS responsibility as associated with data integrity validation. Each is examined to determine whether or not the solution proposed in this SCR sufficiently addresses the area of concern:

- verifying the integrity of data stored on physical media – By requiring the co-location of the checksum file with the data it describes, we ensure that the checksum information will be maintained as long as the data are maintained and are accessible to anyone who has access to an archive volume, whether or not they have immediate access to the PDS node responsible for curating the data. Thus, even if the PDS were to cease to exist, the checksum values would still be available for data validation purposes. This is consistent with the PDS philosophy of having all information necessary for an archive to be self-contained within that archive.

- detecting errors introduced during transfer of data to newer media – As with the above point, the mere existence of the checksum values and their co-location with the data enable the validation of the data during a process of transfer to newer media. (It should be noted that this solution does not address the ability to validate against data corruption during a process of upgrading data product labels to a new version of the PDS standards, when the labels are attached (e.g., converting PDS3 labels to XML-based PDS4). For that, data object checksums would be required. These are outside the scope of this SCR and will be considered in a separate SCR.)

- detecting errors that occur during the transmission of data from data providers to the PDS – The mere requirement for the inclusion of checksum files within archive volumes DOES NOT protect against the introduction of errors during the transmission of data from data providers to the PDS. For this, individual nodes are responsible to negotiate with their data providers a mechanism for ensuring the safe transmission of these data, presumably by the provision of data provider-generated checksums to the receiving node. However, the inclusion of the checksum file as a required file within the archive volumes is likely to serve as an encouragement to at least some data providers to generate this file as a part of their ground data processing stream.

- detecting errors that occur during the transmission of data between PDS nodes – By providing a standardized format and location for the storage of checksum values, we make it much simpler to develop and utilize software that will automatically validate intra-PDS data transfers, whether they are transfers of entire volumes or of single data files.

- detecting errors that occur during the transmission of data from the PDS to the NSSDC - The inclusion of the checksum files within the archive volumes delivered to the NSSDC provides a ready mechanism for ensuring that the data that the NSSDC receives from the PDS is precisely the data that was sent from the PDS (and vice versa, if data ever need to be retrieved from the NSSDC).

- detecting errors that occur during the transmission of data from the PDS to end users – As with intra-PDS transfers, the presence of checksum values in a standardized format and location makes possible the development of a mechanism associated with the product servers for delivering checksum values or data integrity checks to users along with their downloaded products. Note that this does not require users to download the entire checksum file to validate a single file download.

By simply requiring the inclusion of checksum files in archive volumes, this SCR does nothing to require that nodes actively validate their data holdings. There is a whole infrastructure and set of processes that are needed. We have not solved the task of ensuring data integrity, in compliance with our level 3 requirements.

True. Also, as noted above, discipline nodes could comply with the requirements of this SCR by generating checksum files themselves, rather than insisting on their generation by data providers. It is not the function of this SCR to force PDS nodes to fulfill their data validation and maintenance responsibilities.

However, this SCR does specify a standardized mechanism enabling the nodes to perform the required validation and thereby simplifies the accomplishment of those responsibilities. Furthermore, with this standard in place, it then becomes practical for the Management Council, if it so desires, to direct the Engineering Node to devote the necessary resources to develop an infrastructure to ensure data integrity in an automated fashion. (Components of such an infrastructure could include a checksum generation tool, modifications to volume and data product validation tools, modifications to the data product servers, and the generation of a tool to periodically traverse the entire PDS archive and validate data files against their checksum files.) The development of such an infrastructure requires the standardization of the storage of checksum data, and thus justifies, as a precursor, this SCR.

Given that checksum files are just as prone to corruption as any other file in an archive, there appears to be no advantage to storing these files within the archive. They should instead be stored in a separate, safe location (like a database).

There is no such thing as a "safe" location within the data storage world. A database is just as vulnerable to corruption as is an ASCII file stored on a physical disk or online file system. This is the whole point of maintaining backups. By requiring the checksum files to be included within the archive volume, we impose on them the same requirement for multiple distributed copies that we do on the data volumes themselves.

Also, when one compares the relative sizes of an ASCII checksum file with the rest of the data volume it describes, the chances of any minor error being introduced to the checksum file rather than the data are vanishingly small (less than a 10^{th} of a percent, by my back of the envelope calculation on a typical volume such as the MPF IMP EDR CD; obviously the chance would be higher for a volume containing large numbers of extremely small files).

Furthermore, it needs to be kept in mind that the purpose of this SCR is not to restore corrupted data, but merely to detect it. Any mismatch between the contents of a checksum file and a checksum generated on the accompanying data will flag a problem; at that point, further investigation can be carried out by comparing the archive volume with alternate copies to determine where the error lies.

It should also be noted that there is nothing to preclude the storage by a node of checksum values in a database in addition to within the archive volume, if that will facilitate their data repository validation procedures. For those still concerned about the corruption of the checksum files, it should be noted that a checksum value could be generated for each checksum file on each archive volume and stored in a database maintained by the node. This could serve as another level of assurance against data corruption.

Why are we limiting ourselves to a specific checksum algorithm (i.e., MD5) rather than permitting ourselves the option of using other, potentially better algorithms (like SHA-1)?

The MD5 algorithm is completely sufficient for the purposes that the PDS is using it. (Note that while SHA-1 is more secure, it is also slower; we are not in the data encryption business and do not need the level of security provided by SHA-1.) By agreeing on a single checksum algorithm we simplify the requirements on checksum generation and validation tools. This approach is also consistent with the Management Council's desire to simplify, rather than complicate, PDS standards. Nothing in this

SCR precludes us from approving additional checksum algorithms in the future, should they become preferable, after appropriate consideration and review.

By constraining ourselves to a PDS-specific format for the checksum file, we are imposing an unnecessary burden on our data providers, discipline nodes, and end users to use PDS tools to generate and validate data.

The only specification in this proposal for the format of the checksum file is that it be a compliant ASCII TABLE object containing two columns, one containing checksum values and one containing file path specifications. The range of valid options for the representation of the TABLE object within a data file is sufficiently wide to encompass the file format used by most existing checksum utilities. (For example, the example shown in Appendix D is consistent with the md5deep utility.)

Furthermore, if PDS tools are developed to generate and/or validate checksums, they would only serve to make it simpler, not more difficult, for data providers, nodes, and users to do their jobs, by focusing on those PDS-specific aspects of the validation task not handled by off-the-shelf utilities. For example, this might include tasks like:

- automatically generating a CHECKSUM.LBL file to accompany the checksum file
- appending to an existing checksum file if new data are added to an online data repository, and updating the accompanying label
- validating not just a single volume against its checksum file but an entire online archive consisting of multiple volumes, each with its own checksum file
- taking into account the location of the checksum file in the INDEX rather than the ROOT directory
- correcting for upper-/lower-case discrepancies between the contents of a checksum file and the filenames as represented by the file system the data reside on

(Note that the above should not be construed as a set of requirements for any PDS tool, merely as a list of potential capabilities such a tool could have.)

Finally, it is no more difficult for data providers, discipline nodes, and end users to download and install PDS tools than it is for them to download and install off-the-shelf software, so it is unclear what burden is being imposed by the use of PDS tools.

Why are we bothering to concern ourselves with data integrity checks for the transmission of files over the Internet when there already exist built-in error detection and correction mechanisms for these types of file transfers?

This question does not take into account the large and increasing size of individual PDS data products. The chance of a data drop-out in a 5GB image file is huge relative to the to the likelihood of a data drop-out in a much smaller text file. Also, as data transmission rates for these large files become more and more of an issue, it is highly likely that the PDS will need to examine the possibility of utilizing data transport protocols that are more efficient than the traditional TCP (Transmission Control Protocol) that provides the above-mentioned built-in error detection and correction mechanisms. For example, the second most popular method for transferring data over the Internet is UDP (User Datagram Protocol), a much faster data transfer mechanism, but one without the built-in error checking provided by TCP. Thus, the importance of our own mechanisms for verifying the integrity of data transmissions becomes much more important. (More information on these protocols is available at the following URLs: http://en.wikipedia.org/wiki/Transmission_Control_Protocol, http://en.wikipedia.org/wiki/User_Datagram_Protocol.)

By requiring the inclusion of checksum files in archive volumes, we are once again levying new requirements on our current data providers.

While the issue of "when" we start requiring the inclusion of the checksums within PDS archives is not within the scope of this SCR, I will simply point out that as with any other standards change, missions are only required to adhere to the version of the standards that was in place at the time they began their archiving process (i.e., negotiated their contractual obligations with the mission and PDS, wrote archive plans, ICDs, etc.). Therefore, no mission currently providing data to the PDS should be forced to comply with this new standard. (Of course, some may wish to do so voluntarily and there is nothing to prevent them from doing this.)

By requiring the inclusion of checksum files in archive volumes for PDS3 rather than waiting for PDS4, we make it likely that we are going to break validation software (in this case, volume validation software).

By this argument, no changes could be made to the PDS standards until we move to PDS4. With every update we make to the standards, we take the risk of breaking existing software. However, it is the job of our tool developers and maintainers to verify that changes to the standards are implemented in our tools in such a way as to provide for backwards compatibility and for minimal disruption to our data providers currently

submitting data compliant with older standards. This process is done by communicating with the larger PDS community to ensure that tools behave as expected, a task that our existing EN development team actively pursues.

The requirements of this SCR are insufficiently flexible to accommodate the variety of ways in which we receive data from our data providers. In some cases we build volumes; in others they are delivered to our node already completed. Some data sets are delivered in a single batch; others trickle in over a period of multiple years. In the latter case, data are often housed in an online repository for years before being delivered to the NSSDC as a formal archive volume.

Nothing in this SCR precludes the use of checksum "manifests" accompanying data deliveries from data providers to their discipline nodes. The mechanism for ensuring the integrity of the data delivery from the provider to the node is the responsibility of the node and should be negotiated between them and their data providers. Neither does this SCR preclude the use of checksum files within online repositories, whether they are formally archived or not. It would be a trivial matter to append to a checksum file for such an online virtual volume every time a new data delivery was made. Again, it is the responsibility of the node to utilize the checksum information available to them to verify the integrity of their data; it is not the purpose of a standards change request to mandate this.

We feel that maintaining a checksum file on a volume would be a bigger task than most realize. This would be particularly true for missions that periodically deliver data over a long duration or many extended missions. It would also be a burden for missions where data providers revise and redeliver all or some of their products.

It is not clear how the duration of a mission should impact the difficulty of maintaining checksum files in compliance with this SCR. Whether the data for the mission are delivered in "chunks" comprising multiple independent volumes or whether the data are aggregated in an online repository as a single large virtual volume should have no impact on the generation or maintenance of the checksum file(s); all updates to and validation against the checksum files should be handled by automated scripts which would require very little effort to write assuming the format and location of the checksum files are standardized (which is what is proposed in this SCR).

As to the issues of data providers re-delivering their data products – assuming the data providers have changed either the data products, their labels, or both (which they presumably have, since they're re-delivering), the previously generated checksums

become useless. If the re-delivery constitutes an entire volume, it should require no more effort to generate new checksums than it took to generate them in the first place (i.e. virtually no effort). The only complication could arise if only scattered products within a volume are re-delivered; in this case, multiple approaches could be taken. A manifest of the updated files could be delivered along with the files and used as input to a script which would generate new checksums for those files and update the appropriate records within the checksum file. Alternatively, after the updated files are delivered, a validation run against the original checksum file could be run. This should produce a list of errors (i.e. files whose checksum doesn't match that in the checksum file); this list could then be checked to ensure that all the files on it are newly delivered files and the checksum file would be updated. Thirdly, since the node has presumably received checksum values for the updated files from the data provider as part of their delivery, the checksum values could be taken directly from the data provider and parsed into the existing checksum file by an automated script. Then after the new data files are emplaced in the volume, a validation run could be made against the updated checksum file. No matter how this is handled, these tasks should be easily automated with a simple script and should not present any serious hurdles to compliance with this SCR. Standardization is the key in all these cases to simplifying the task at hand.

We do not deliver volumes to users in most cases. Most of the time users download individual products. If a volume contains many thousands of products, then I cannot see users also downloading the checksum file and parsing it to find the checksum value for products of interest.

It is not expected that end users would download an entire checksum file to look up a single data file's checksum. A more sensible method for the validation of individual data product transfers would be to include a mechanism within the data product servers to automatically look up and validate against the checksum for an individual requested data product. However, the implementation of this kind of mechanism requires a standardized storage format and location for the checksum information, and thus justifies this SCR as a precursor.

```
PDS_VERSION_ID                    = PDS3
LABEL_REVISION_NOTE               = "2004-04-06, CN: BAM;
2004-10-14, PPI: S. Joy; 2006-07-11, EN: EDR"

OBJECT                            = ELEMENT_DEFINITION
  ELEMENT_NAME                    = "md5_checksum"
  BL_NAME                         = "md5checksum"
  DESCRIPTION                     = "

   The MD5 algorithm takes as input a file (message) of arbitrary length
   and produces as output a 128-bit 'fingerprint' or 'message digest' of
   the input. It is conjectured that it is computationally infeasible to
   produce two messages having the same message digest, or to produce
   any message having a given prespecified target message digest.

   Most standard MD5 checksum calculators return a 32 character
   hexadecimal value containing lower case letters.  In order to accommodate
   this existing standard, the PDS requires that the value assigned to the
   MD5_CHECKSUM keyword be a value composed of lowercase letters (a-f) and
   numbers (0-9).  In order to comply with other standards relating to the
   use of lowercase letters in strings, the value must be quoted using double
   quotes.

   Example:  MD5_CHECKSUM = "0ff0a5dd0f3ea4e104b0eae98c87f36c"

   The MD5 algorithm was described by its inventor, Ron Rivest of
   RSA Data Security, Inc., in an Internet Request For Comments
   document, RFC1321 (document available from the PDS).

   References
   ==========
   Rivest, R., The MD5 Message-Digest Algorithm, RFC 1321, MIT Laboratory for
   Computer Science and RSA Data Security, Inc., April 1992."

  GENERAL_DATA_TYPE               = "CHARACTER"
  MAXIMUM                         = ""
  MINIMUM                         = ""
  MAXIMUM_LENGTH                  = "32"
  MINIMUM_LENGTH                  = "32"
  STANDARD_VALUE_TYPE             = "DEFINITION"
  STANDARD_VALUE_SET_DESC         = "N/A"
  KEYWORD_DEFAULT_VALUE           = "N/A"
  UNIT_ID                         = "NONE"
  SOURCE_NAME                     = "PDS CN/B. SWORD"
  FORMATION_RULE_DESC             = "N/A"
  SYSTEM_CLASSIFICATION_ID        = "COMMON"
  GENERAL_CLASSIFICATION_TYPE     = "N/A"
  CHANGE_DATE                     = "2006-03-20"
  STATUS_TYPE                     = "APPROVED"
  STANDARD_VALUE_OUTPUT_FLAG      = "N"
  TEXT_FLAG                       = "N"
  TERSE_NAME                      = "md5checksum"
  SQL_FORMAT                      = "CHAR(32)"
  BL_SQL_FORMAT                   = "char(32)"
  DISPLAY_FORMAT                  = "JUSTLEFT"
  AVAILABLE_VALUE_TYPE            = "N/A"
END_OBJECT                        = ELEMENT_DEFINITION
END
```

```
PDS_VERSION_ID                  = PDS3
LABEL_REVISION_NOTE             = "2006-07-11, EN: EDR"

OBJECT                          = ELEMENT_DEFINITION
  ELEMENT_NAME                  = "checksum_type"
  BL_NAME                       = "checksumtype"
  DESCRIPTION                   = "
   The CHECKSUM_TYPE keyword is used to specify the type of checksum
   algorithm used to calculate a checksum for a file or data object."


  GENERAL_DATA_TYPE             = "IDENTIFIER"
  MAXIMUM                       = "N/A"
  MINIMUM                       = "N/A"
  MAXIMUM_LENGTH                = "12"
  MINIMUM_LENGTH                = "1"
  STANDARD_VALUE_TYPE           = "DYNAMIC"
  STANDARD_VALUE_SET            = {"MD5"}
  STANDARD_VALUE_SET_DESC       = "N/A"
  KEYWORD_DEFAULT_VALUE         = "N/A"
  UNIT_ID                       = "N/A"
  SOURCE_NAME                   = "PDS EN/E. RYE"
  FORMATION_RULE_DESC           = "N/A"
  SYSTEM_CLASSIFICATION_ID      = "COMMON"
  GENERAL_CLASSIFICATION_TYPE   = "N/A"
  CHANGE_DATE                   = "2006-03-20"
  STATUS_TYPE                   = "APPROVED"
  STANDARD_VALUE_OUTPUT_FLAG    = "Y"
  TEXT_FLAG                     = "N"
  TERSE_NAME                    = "checksumtype"
  SQL_FORMAT                    = "CHAR(12)"
  BL_SQL_FORMAT                 = "char(12)"
  DISPLAY_FORMAT                = "JUSTLEFT"
  AVAILABLE_VALUE_TYPE          = "N/A"
END_OBJECT                      = ELEMENT_DEFINITION
END
```

D.2 CHECKSUM.TAB and CHECKSUM.LBL

Each PDS archive volume must include a "CHECKSUM.TAB" file in the INDEX subdirectory. This file must be accompanied by a detached PDS label. The CHECKSUM.TAB file contains a checksum for every file contained within the archive volume (or in the entire archive, if stored as a virtual volume online), with the exception of the checksum file itself and its label.

A CHECKSUM.TAB file is a PDS ASCII TABLE object comprising two required COLUMN objects. One COLUMN object has the name CHECKSUM; the other has the name FILE_SPECIFICATION_NAME. The definition of the CHECKSUM column must include the keyword-value pair "CHECKSUM_TYPE = MD5".

D.2.1 Example of CHECKSUM.TAB

```
1e8d45f622e09b9e2998af1a6d67a296 AAREADME.TXT
7dcfa51691ddd149a5a091ebe87b9bb1 ERRATA.TXT
f8dd7758cb5231c9e7817c4710d00b6e BROWSE/MARS/C1246XXX/I862934L.IMG
8ed31a70bc95aa104edbf4bc30a8c199 BROWSE/MARS/C1246XXX/I862934L.LBL
d8b83365f5e117b9665181944889da3d BROWSE/MARS/C1246XXX/I862934R.IMG
.
.
.
```

D.2.2 Example of CHECKSUM.LBL

```
PDS_VERSION_ID               = PDS3

RECORD_TYPE                  = FIXED_LENGTH
RECORD_BYTES                 = 71
FILE_RECORDS                 = 3623

DESCRIPTION                  = "CHECKSUM.TAB provides a checksum for all
                                files included on this archive volume, with
                                the exception of the checksum file itself
                                and its label."

^CHECKSUM_TABLE              = "CHECKSUM.TAB"

OBJECT                       = CHECKSUM_TABLE
  INTERCHANGE_FORMAT         = ASCII
  ROW_BYTES                  = 71
  ROWS                       = 3623
  COLUMNS                    = 2

  OBJECT                     = COLUMN
    NAME                     = CHECKSUM
    DESCRIPTION              = "The checksum of the indicated file."
```

```
    CHECKSUM_TYPE           = MD5
    DATA_TYPE               = CHARACTER
    START_BYTE              = 1
    BYTES                   = 32
  END_OBJECT                = COLUMN

  OBJECT                    = COLUMN
    NAME                    = FILE_SPECIFICATION_NAME
    DESCRIPTION             = "Identifies the file for which the checksum
                               was calculated."
    DATA_TYPE               = CHARACTER
    START_BYTE              = 34
    BYTES                   = 36
  END_OBJECT                = COLUMN

END_OBJECT                  = CHECKSUM_TABLE
END
```