


ReplicationTaskSuite (WIP)

This Document is a Work In Progress


 NOTICE: This is a COPY of the ReplicationTaskSuite document, currently being used to document a change that is not yet available.

Replication Task Suite

The Replication Task Suite is a DSpace Add-On which provides a set of [curation system tasks](#) to assist in performing replication (backup/restore/audit) of DSpace contents to other locations. The DSpace content is packaged in containers known as AIPs ([OAIS](#) speak: 'archival information packages'). By default, AIPs are generated in the default [DSpace AIP Format](#) (the same format used by the [AIP Backup and Restore](#) tool). If desired, there is an option to generate [BagIt-based](#) AIPs instead of using the default DSpace AIP format.


This Add-On also integrates DSpace with DuraCloud for users that wish to easily back up their content into DuraCloud directly from their DSpace administrative interface.

Latest Releases of Replication Task Suite


 Based on the version of DSpace you are running, here are the compatible latest releases of the Replication Task Suite:

- **RTS, version 6.0** - bug-fix release, compatible only with DSpace 6.x and higher releases
 - *Upgrading:* To upgrade to RTS 6.0 from a previous version, simply change your pom.xml (see [Installation on DSpace 6.x](#)) to reference 'dspace-replicate' version 6.0. Then rebuild DSpace & re-run 'ant update'. You should verify your configurations are still compatible with DSpace 6.0, as the DSpace [Configuration System](#) received an overhaul in DSpace 6
 - After upgrading the RTS software, it is recommended to run a full backup to ensure all your AIP packages are also updated (if necessary).
 - **6.0 Bug Fixes:** refactored to work with DSpace 6.x (UUIDs instead of integers)
- **RTS, version 3.4** - bug-fix release, compatible with all DSpace 3.x, 4.x and 5.x releases
 - *Upgrading:* To upgrade to RTS 3.4 from a previous version, simply change your pom.xml (see [Installation on DSpace 3.x, 4.x or 5.x](#)) to reference 'dspace-replicate' version 3.4. Then rebuild DSpace & re-run 'ant update'. Your existing RTS 3.x configuration files will still work with RTS 3.4.
 - After upgrading the RTS software, it is recommended to run a full backup to ensure all your AIP packages are also updated (if necessary).
 - **3.4 Bug Fixes:** Updates DuraCloud API to v4.3.9 (to take advantage of recent performance improvements in DuraCloud API). **WARNING: The DuraCloud API v4.0.0 or greater requires Java 8. So, using RTS 3.4 with DuraCloud would require running DSpace with Java 8.**
 - **3.3 Bug Fixes:** Updates DuraCloud API to v3.7.4.
 - **3.2 Bug Fixes:** Ensures RTS compiles for Java 7 (which is required for RTS 3.x).
 - **3.1 Bug Fixes:** Fixes several small bugs (namely with the event consumer utilized during [Automatic Replication](#)). Also ensures RTS runs on the DSpace 3.x platform.
- **RTS, version 1.3** - bug-fix release, compatible with all DSpace 1.8.x releases.
 - *Upgrading:* To upgrade to RTS version 1.3 from a previous release, simply change your pom.xml (see [Installation on DSpace 1.8.x](#)) to reference 'dspace-replicate' version 1.3. Then rebuild DSpace & re-run 'ant update'. Your existing RTS 1.x configuration files will still work with RTS 1.3.
 - After upgrading the RTS software, it is recommended to run a full backup to ensure all your AIP packages are also updated (if necessary).
 - **1.3 Bug Fixes:** This fixes a DuraCloud v2.4.0 connection error with version 1.2.
 - **1.2 Bug Fixes:** This fixes a Java 6 incompatibility bug in version 1.1. Previously version 1.1 required Java 7 when using DuraCloud.
 - **1.1 Bug Fixes:** Fixes for several small bugs in 1.0 (namely with the event consumer utilized during [Automatic Replication](#)).

Usage Examples

 For a quick overview of the various tasks offered in the Replication Task Suite, along with some real-life scenarios / examples of where each Replication task may come in handy, you may wish to skip directly to the [Problem Statement and Usage Examples](#) section at the bottom of this page.

More Information / Screenshots

 More information on the Replication Task Suite is available from the following webinars / screencasts:

- 2013-Apr-24: [Preserving DSpace Content with DuraCloud](#) (DuraCloud Brown Bag Series)
- 2011-Dec-1: [Screen cast of DSpace back up to DuraCloud using Replication Task Suite](#)
- 2011-Nov-30: [Integrating DSpace with DuraCloud \(using Replication Task Suite\) Webinar](#) (slides + recorded audio)
- 2011-Jun-7: [DuraCloud Workshop at Open Repositories 2011](#) (slides only)

1 Installation

- 1.1 [Supported DSpace and Java Versions](#)
 - 1.1.1 [User Interface Compatibility Notes](#)
- 1.2 [Installation on DSpace 6.x](#)
- 1.3 [Installation on DSpace 3.x, 4.x or 5.x](#)
- 1.4 [Installation on DSpace 1.8.x](#)

2 Upgrades

3 Configuration

- 3.1 [Enabling Replication Task Suite](#)
- 3.2 [Overview of Configuration Options](#)
- 3.3 [AIP Format Options](#)

3.3.1	Configuring usage of DSpace default AIP Format (METS-based)
3.3.2	Configuring usage of DSpace BagIt AIP Format
3.4	Storage Options
3.4.1	Configuring Local Storage
3.4.2	Configuring Mountable Storage
3.4.3	Configuring DuraCloud Storage
3.4.3.1	How DuraCloud storage works
3.4.3.2	DuraCloud Account Settings
3.4.3.3	DuraCloud Storage Settings
3.5	Automation Options (Recommended)
3.5.1	Automatically Sync Changes (via Queue)
3.5.1.1	Activate the Sync Consumer
3.5.1.2	How the Sync Consumer works
3.5.1.3	Configuring the Sync Consumer
3.5.1.4	Processing the Sync Consumer Queue
3.5.1.5	Enhancing the Performance of the Queue Processing (optional)
3.5.2	Scheduled Site Auditing/Replication
3.6	Additional Options
3.6.1	Configuring usage of Checkm manifest validation
4	Problem Statement and Usage Examples
4.1	First Steps - Estimation
4.2	Replicating
4.3	Verifying Replication
4.4	Ensuring Replica Integrity and Accuracy over time
4.5	Repairing Damage
4.5.1	Restoring Object(s)
4.5.2	Replacing Object(s)
4.6	Cleanup
4.7	Keeping Score
4.8	Automation (Recommended)
4.9	Replica Storage / Backup Location
5	Codebase / Development

Installation

Supported DSpace and Java Versions

The Replication Task Suite currently supports the following versions of DSpace software:

Replication Task Suite Version	Supported DSpace Version(s)	Supported Java Version	Supported Interfaces	Notes
6.0	DSpace version 6.x or higher	Java 7 or above	XMLUI and/or commandline	The 6.0 stable version of the Replication Task Suite offers no new functionality over the previous versions. It is simply a refactor of the code to ensure that Replication Task Suite works with DSpace 6.x and later versions - see DS-3389
3.4	DSpace version 3.x, 4.x or 5.x	Java 7 or above	XMLUI and/or commandline	The 3.4 stable version of the Replication Task Suite is nearly identical to the 1.x stable version. It just includes minor bug fixes to ensure the Replication Task Suite is compatible with the newer DSpace APIs.
1.3	DSpace version 1.8.x	Java 6 or above	XMLUI and/or commandline	Highly recommended to use either DSpace 1.8.1 or above. DSpace 1.8.0 has a known bug where running a Replication Task will always return a NullPointerException - see DS-1077

Installation instructions for each version are included below:

- [Installation on DSpace 6.x](#)
- [Installation on DSpace 3.x, 4.x or 5.x](#)
- [Installation on DSpace 1.8.x](#)

User Interface Compatibility Notes

As the Replication Suite is just a suite of [Curation System](#) tasks, it may be called (like any Curation Tasks) from the following locations:

- From the Command Line
- From the Admin UI (XMLUI Only)
- From Item Approval Workflow
- From custom Java code

For more information see the Curation System details on [Task Invocation](#).

Installation on DSpace 6.x

1. In your DSpace Source directory ([`dspace-src`]), you will need to modify the following POM file:
 - [`dspace-src`]/`dspace/modules/additions/pom.xml` (This POM will ensure that the "dspace-replicate" dependency is made available to commandline and ALL DSpace interfaces)

2. For this pom.xml file, add the following <dependency> section at the end of the existing <dependencies> section (just before the closing </dependencies> tag). NOTE: the exclusions are *required* to work around [DS-3536](#).

```
<dependencies>
  ...

  <!-- Adding this dependency will install the Replication Task Suite Addon -->
  <dependency>
    <groupId>org.dspace</groupId>
    <artifactId>dspace-replicate</artifactId>
    <version>6.0</version>
    <!-- These exclusions are currently necessary to resolve dependency mismatches with some
dependencies pulled into RTS 6.0 to work with DuraCloud, see DS-3536 for details -->
    <exclusions>
      <exclusion>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
      </exclusion>
      <exclusion>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-java-sdk-core</artifactId>
      </exclusion>
      <exclusion>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpmime</artifactId>
      </exclusion>
      <exclusion>
        <groupId>org.springframework</groupId>
        <artifactId>spring-expression</artifactId>
      </exclusion>
      <exclusion>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-core</artifactId>
      </exclusion>
      <exclusion>
        <groupId>org.codehaus.jackson</groupId>
        <artifactId>jackson-mapper-asl</artifactId>
      </exclusion>
      <exclusion>
        <groupId>org.codehaus.jackson</groupId>
        <artifactId>jackson-core-asl</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

3. Once you've finished modifying both pom.xml files, rebuild DSpace by running the following from your [dspace-src]/dspace/ folder:

```
mvn clean package
```

4. Update the default dspace.cfg to include the Replication Task Suite config files. This ensures these configs are loaded as part of your DSpace configuration. This also allows you to override the configurations in your own local.cfg file.

```
include = ${module_dir}/replicate.cfg
include = ${module_dir}/replicate-mets.cfg
```

- a. You should ensure these configurations exist in your [dspace-src]/dspace/config/ directory. That way they will be auto-installed /copied whenever you run "ant update" (see next step).
5. Follow the instructions in the [Configuration](#) section below in order to enable & configure the Replication Task Suite Add-On.
6. You will need to update your existing DSpace 3.x installation, by running the following from your [dspace-src]/dspace/target/dspace-[version]-build/ directory

```
ant update
```

Alternatively, if you don't want to do a full DSpace update, you can just update your existing binaries & webapps by running the following two commands:

- `ant update_code` (Updates the existing `[dspace]/lib/` directory)
- `ant update_webapps` (Updates the existing `[dspace]/webapp/` directory)

Installation on DSpace 3.x, 4.x or 5.x

1. In your DSpace Source directory (`[dspace-src]`), you will need to modify the following POM file:
 - `[dspace-src]/dspace/modules/additions/pom.xml` (This POM will ensure that the "dspace-replicate" dependency is made available to commandline and ALL DSpace interfaces)
2. For this `pom.xml` file, add the following `<dependency>` section at the end of the existing `<dependencies>` section (just before the closing `</dependencies>` tag).

```
<dependencies>
...

    <!-- Adding this dependency will install the Replication Task Suite Addon -->
    <dependency>
        <groupId>org.dspace</groupId>
        <artifactId>dspace-replicate</artifactId>
        <version>3.4</version>
    </dependency>
</dependencies>
```

3. Once you've finished modifying both `pom.xml` files, rebuild DSpace by running the following from your `[dspace-src]/dspace/` folder:

```
mvn clean package
```

4. Follow the instructions in the [Configuration](#) section below in order to enable & configure the Replication Task Suite Add-On.
 - a. You may wish to ensure these configurations exist in your `[dspace-src]/dspace/config/` directory. That way they will be auto-installed/copied whenever you run "ant update" (see next step).
5. You will need to update your existing DSpace 3.x installation, by running the following from your `[dspace-src]/dspace/target/dspace-[version]-build/` directory


```
ant update
```

Alternatively, if you don't want to do a full DSpace update, you can just update your existing binaries & webapps by running the following two commands:

- `ant update_code` (Updates the existing `[dspace]/lib/` directory)
- `ant update_webapps` (Updates the existing `[dspace]/webapp/` directory)

Installation on DSpace 1.8.x

Known Curation System bug in 1.8.0

 DSpace 1.8.0 contains a bug in the Curation System which causes a `NullPointerException` error to be returned when any curation task is run across the entire site (see [DS-1077](#)). This bug directly affects the Replication Task Suite. Even when a replication task succeeds, it will still throw a `NullPointerException`. You can check the DSpace logs to tell whether the task actually succeeded or not. This bug was resolved in DSpace 1.8.1.

Because of the above bug, we recommend running the Replication Task Suite on DSpace 1.8.1 or above.

1. In your DSpace Source directory (`[dspace-src]`), you will modify two Maven `pom.xml` files:
 - `[dspace-src]/dspace/pom.xml` (This POM controls dependencies of CommandLine scripts. Modifying it will let you run `dspace-replicate` from commandline)
 - `[dspace-src]/dspace/modules/xmlui/pom.xml` (This POM controls dependencies of XMLUI. Modifying it will let you run `dspace-replicate` from XMLUI)
2. For **each** of these `pom.xml` files, add the following `<dependency>` section at the end of the existing `<dependencies>` section (just before the closing `</dependencies>` tag).

```
<dependencies>
  ...

  <!-- Adding this dependency will install the Replication Task Suite Addon -->
  <dependency>
    <groupId>org.dspace</groupId>
    <artifactId>dspace-replicate</artifactId>
    <version>1.3</version>
  </dependency>
</dependencies>
```

- Once you've finished modifying both pom.xml files, rebuild DSpace by running the following from your [dspace-src]/dspace/ folder:

```
mvn clean package
```

- Follow the instructions in the [Configuration](#) section below in order to enable & configure the Replication Task Suite Add-On.
 - You may wish to ensure these configurations exist in your [dspace-src]/dspace/config/ directory. That way they will be auto-installed/copied whenever you run "ant update" (see next step).
- You will need to update your existing DSpace 1.8.x installation, by running the following from your [dspace-src]/dspace/target/dspace-[version]-build/ directory

```
ant update
```

Alternatively, if you don't want to do a full DSpace update, you can just update your existing binaries & webapps by running the following two commands:

- ant update_code (Updates the existing [dspace]/lib/ directory)
- ant update_webapps (Updates the existing [dspace]/webapp/ directory)

Upgrades

Upgrading the Replication Task Suite to a new version essentially involves a reinstallation of the add-on.

Follow the latest installation instructions, based on the version of DSpace you are running:

- [Installation on DSpace 6.x](#)
- [Installation on DSpace 3.x, 4.x or 5.x](#)
- [Installation on DSpace 1.8.x](#)

Once you have reinstalled the Replication Task Suite, you should compare your existing configurations with the latest Replication Task Suite configurations. In most cases, your existing configurations should function perfectly, but you should review the differences just in case.

Configuration

Configuration of the Replication Task Suite is based entirely on your local institution's backup, restore and preservation needs.

Enabling Replication Task Suite

In order to enable the Replication Task Suite, you need to create / edit several configuration files.


- A copy of all configuration files utilized by the Replication Task Suite (RTS) can be found in the following locations:
 - Configs for RTS version 1.x : https://github.com/DSpace/dspace-replicate/tree/dspace-replicate-1_x/config/modules
 - Configs for RTS version 3.x : https://github.com/DSpace/dspace-replicate/tree/dspace-replicate-3_x/config/modules
 - Configs for RTS version 6.x : <https://github.com/DSpace/dspace-replicate/tree/master/config/modules>
- Copy the following configuration files to your DSpace's [dspace]/config/modules/ directory:
 - replicate.cfg - This file contains the base settings for the Replication Task Suite
 - replicate-mets.cfg - This file provides a few additional replication options specific to METS-based AIPs (see below for more details)
 - duracloud.cfg - If you'd like to replicate/backup your content to [DuraCloud](#), this file holds your DuraCloud account information
- Edit your [dspace]/config/modules/curate.cfg configuration file to define & enable all tasks. The list of tasks to add to this configuration file depends on which type of AIP (METS based or BagIt based) you wish to use. Please see the [AIP Format Options](#) section below for the details of what should be added to your curate.cfg file
 - A sample, fully enabled curate.cfg configuration file is provided alongside the other Replication Task Suite config files listed above. This sample file is preconfigured to use METS-based AIPs.
- Recommended (but not required): Edit your [dspace]/config/modules/dspace.cfg and enable the Replication Task Suite 'listener' to perform automatic synchronization of your AIP backup store with what is in DSpace (see [Automation Options](#) for more info).

Overview of Configuration Options

Before getting started, you may wish to determine the answers to the following questions:

1. [AIP Format Options](#): Does your institution want to backup using the default DSpace AIP format (METS packaging)? Or would you rather utilize the new BagIt AIP Format?
2. [Storage Options](#): Does your institution plan to use the Replication Suite to backup to a local/mounted drive? Or would you like to connect it to a DuraCloud account?
3. [Automation Options \(Recommended\)](#): Do you want to automatically sync your AIP backup store with what is in DSpace? (*this is highly recommended, but not required*)
4. [Additional Options](#): Do you plan to use Checkm manifests for checksum auditing?

Overview of Task Suite usage

 For a higher level introduction to the Replication Task Suite, please see the [Problem Statement and Usage Examples](#) section below. It may provide you with a better idea of how you'd like to configure this task suite based on your institutional needs.

AIP Format Options

One of the first questions to ask yourself is the format you wish to utilize for your AIPs.

There are two options:

1. [DSpace AIP Format \(METS-based\)](#) (default) - This is the same [AIP format](#) utilized by the DSpace [AIP Backup and Restore](#) feature, so it is 100% compatible with that DSpace feature. In fact when using this format, the Replication Task Suite just "wraps" calls to the [AIP Backup and Restore](#) feature itself.
2. [BagIt AIP Format](#) (beta) - This is a new AIP format provided by the Replication Task Suite. It generates AIPs in the [BagIt File Packaging Format](#). Institutions which already are familiar with BagIt or use it elsewhere may find this format preferable. (Please note that this AIP format does not yet support all DSpace objects. See the below table for more information.)

These two AIP formats are not identical. The below table seeks to describe some of the differences.

	DSpace AIP Format (METS-based AIPs)	BagIt AIP Format
Supported Backup/Restore Types		
Can Backup & Restore all DSpace Content easily	Yes	Yes
Can Backup & Restore a Single Community /Collection/Item easily	Yes	Yes
Backups can be used to move one or more Community/Collection/Items to another DSpace system easily.	Yes (Using the Replication Task Suite or using the command line AIP Backup and Restore tools)	Yes (though the Replication Task Suite add-on must be installed on both systems)
Can Backup & Restore Item Versions (added in DSpace 3.x)	No (Item Versioning not yet compatible with AIP format. Only the most recent version of an Item is described in the AIP.)	No (Item Versioning not yet compatible with AIP format. Only the most recent version of an Item is described in the AIP.)
Supported DSpace Object Types		
Supports backup/restore of all Communities /Collections/Items (including metadata, files, logos, etc.)	Yes	Yes
Supports backup/restore of all People/Groups /Permissions	Yes	No (Not yet supported)
Supports backup/restore of all Collection-specific Item Templates	Yes	No (Not yet supported)
Supports backup/restore of all Collection Harvesting settings (only for Collections which pull in all Items via OAI-PMH or OAI-ORE)	No (The harvest settings are not preserved, but previously harvested items are preserved in their own AIPs)	No (The harvest settings are not preserved, but previously harvested items are preserved in their own AIPs)
Supports backup/restore of all Withdrawn (but not deleted) Items	Yes	Yes
Supports backup/restore of Item Mappings between Collections	Yes	Yes
Supports backup/restore of all in-process, uncompleted Submissions (or those currently in an approval workflow)	No (AIPs are only generated for objects which are completed and considered "in archive")	No (AIPs are only generated for objects which are completed and considered "in archive")
Supports backup/restore of Items using custom Metadata Schemas & Fields	Yes	Yes

Supports backup/restore of all local DSpace Configurations and Customizations	No (You are expected to backup your DSpace configurations and customizations separately. AIPs only backup content held within DSpace.)	No (You are expected to backup your DSpace configurations and customizations separately. AIPs only backup content held within DSpace.)
---	---	---

For more information on the tasks available based on your AIP format choice, please see the [Problem Statement and Usage Examples](#) section below. This section also provides good examples of how to use each of the tasks available to you in the Replication Task Suite.

Configuring usage of DSpace default AIP Format (METS-based)

This section goes through the steps of configuring the Replication Suite to use the default [DSpace AIP format](#), which utilizes METS packaging. This is the default & recommended setting.

1. **General Curation Configuration:** First, in your `[dspace]/config/modules/curate.cfg` you will want to enable & configure the METS-based replication tasks. (NOTE: there is a sample `curate.cfg` file provided in <https://github.com/DSpace/dspace-replicate/tree/master/config/modules> which is pre-configured to use METS-based AIPs).
 - **Enable the Replication Tasks:** In the list of "Task Class implementations" (`plugin.named.org.dspace.curate.CurationTask`), add the following.
REMEMBER to add a comma and backslash (" , \") after each line (except the final line).

```
plugin.named.org.dspace.curate.CurationTask = \
... (YOUR EXISTING TASKS) ... , \
org.dspace.ctask.replicate.EstimateAIPSize = estaipsize, \
org.dspace.ctask.replicate.ReadOdometer = readodometer, \
org.dspace.ctask.replicate.TransmitAIP = transmitaip, \
org.dspace.ctask.replicate.TransmitSingleAIP = transmitsingleaip, \
org.dspace.ctask.replicate.VerifyAIP = verifyaip, \
org.dspace.ctask.replicate.FetchAIP = fetchaip, \
org.dspace.ctask.replicate.CompareWithAIP = auditaip, \
org.dspace.ctask.replicate.RemoveAIP = removeaip, \
org.dspace.ctask.replicate.METSRestoreFromAIP = restorefromaip, \
org.dspace.ctask.replicate.METSRestoreFromAIP = replacewithaip, \
org.dspace.ctask.replicate.METSRestoreFromAIP = restorekeepexisting, \
org.dspace.ctask.replicate.METSRestoreFromAIP = restoresinglefromaip, \
org.dspace.ctask.replicate.METSRestoreFromAIP = replacesinglewithaip
```

- **Give Each Task a Human-Friendly Task Name:** Under the `ui.tasknames` setting, give each of the above Tasks a human-friendly name. Here are some recommended values, but you are welcome to tweak them.
REMEMBER to add a comma and backslash (" , \") after each line (except the final line).

```
ui.tasknames = \
... (YOUR EXISTING TASK NAMES) ... , \
estaipsize = Estimate Storage Space for AIP(s), \
readodometer = Read Odometer, \
transmitaip = Transmit AIP(s) to Storage, \
verifyaip = Verify AIP(s) exist in Storage, \
fetchaip = Fetch AIP(s) from Storage, \
auditaip = Audit against AIP(s), \
removeaip = Remove AIP(s) from Storage, \
restorefromaip = Restore Missing Object(s) from AIP(s), \
replacewithaip = Replace Existing Object(s) with AIP(s), \
restorekeepexisting = Restore Missing Object(s) but Keep Existing Objects, \
restoresinglefromaip = Restore Single Object from AIP, \
replacesinglewithaip = Replace Single Object with AIP
```

- **Optionally Create a Task Group:** Finally, if you'd like to create a Task Group for these tasks, you can create a group named "replicate" and add them all to it. The below is just an example of how you may wish to set the `ui.taskgroups` and `ui.taskgroup.*` settings. It creates two Task Groups: (1) a "General Purpose Tasks" group for a few default DSpace Curation Tasks, and (2) a "Replication Suite Tasks" group for all these new Replication tasks.

```
# Tasks may be organized into named groups which display together in UI drop-downs
ui.taskgroups = \
    general = General Purpose Tasks, \
    replicate = Replication Suite Tasks

# Group membership is defined using comma-separated lists of task names, one property per group
ui.taskgroup.general = profileformats, requiredmetadata, checklinks
ui.taskgroup.replicate = estaipsize, readodometer, transmitaip, verifyaip, fetchaip, auditaip,
removeaip, restorefromaip, replacewithaip, restorekeepexisting, restoresinglefromaip,
replacesinglewithaip
```

2. **Replication Suite Configuration:** Next, in your `[dspace]/config/modules/replicate.cfg` you will want to ensure it is setup to properly use METS-based AIPs. Under the "AIP Packaging Settings" you'll want the following settings enabled:

```
# Package type. Permitted values: 'mets', 'bagit'
# mets = Generate default DSpace AIPs as described in: https://wiki.duraspace.org/display/DSDOC18/AIP+Backup+and+Restore
# bagit = Generate AIPs based on the BagIt packaging format: https://wiki.ucop.edu/display/Curation/BagIt
packer.pkgtype = mets

# Format of package compression. Permitted values: 'zip' or 'tgz'
# for 'mets' packages, only 'zip' is supported
packer.archfmt = zip

# Whether or not the name packages with a DSpace type prefix.
# When 'true', package files are named [type]@[handle].[format] (e.g. ITEM@123456789-1.zip)
# When 'false', package files are named [handle].[format] (e.g. 123456789-1.zip)
# Defaults to 'true'. For 'mets' packages, this must be 'true'.
packer.typeprefix = true
```

3. **Optionally tweak the AIP Restore/Replace settings:** Optionally, you can decide to tweak the way AIPs are restored or replaced (using [AIP Backup and Restore](#) options). These settings normally **should not need to be tweaked**, but are available in the `[dspace]/config/modules/replicate-mets.cfg` configuration file. See that configuration file for more details.

Configuring usage of DSpace BagIt AIP Format

This section goes through the steps of configuring the Replication Suite to use BagIt-based AIPs. For more information on the BagIt packaging format, see: <https://wiki.ucop.edu/display/Curation/BagIt>

1. **General Curation Configuration:** First, in your `[dspace]/config/modules/curate.cfg` you will want to enable & configure the BagIt-based replication tasks. (NOTE: there is a sample `curate.cfg` file provided in <https://github.com/DSpace/dspace-replicate/tree/master/config/modules> which provides example settings, though they are all commented out by default).
 - **Enable the Replication Tasks:** In the list of "Task Class implementations" (`plugin.named.org.dspace.curate.CurationTask`), add the following.
REMEMBER to add a comma and backslash (", \") after each line (except the final line).

```
plugin.named.org.dspace.curate.CurationTask = \
    ... (YOUR EXISTING TASKS) ... , \
    org.dspace.ctask.replicate.EstimateAIPSize = estaipsize, \
    org.dspace.ctask.replicate.ReadOdometer = readodometer, \
    org.dspace.ctask.replicate.TransmitAIP = transmitaip, \
    org.dspace.ctask.replicate.VerifyAIP = verifyaip, \
    org.dspace.ctask.replicate.FetchAIP = fetchaip, \
    org.dspace.ctask.replicate.CompareWithAIP = auditaip, \
    org.dspace.ctask.replicate.RemoveAIP = removeaip, \
    org.dspace.ctask.replicate.BagItRestoreFromAIP = restorefromaip, \
    org.dspace.ctask.replicate.BagItReplaceWithAIP = replacewithaip
```

- **Give Each Task a Human-Friendly Task Name:** Under the `ui.tasknames` setting, give each of the above Tasks a human-friendly name. Here are some recommended values, but you are welcome to tweak them.
REMEMBER to add a comma and backslash (", \") after each line (except the final line).


```

ui.tasknames = \
... (YOUR EXISTING TASK NAMES) ... , \
estaipsize = Estimate Storage Space for AIP(s), \
readodometer = Read Odometer, \
transmitaip = Transmit AIP(s) to Storage, \
verifyaip = Verify AIP(s) exist in Storage, \
fetchaip = Fetch AIP(s) from Storage, \
auditaip = Audit/Compare against AIP(s), \
removeaip = Remove AIP(s) from Storage, \
restorefromaip = Restore Missing Object(s) from AIP(s), \
replacewithaip = Replace Existing Object(s) with AIP(s)

```

- **Optionally Create a Task Group:** Finally, if you'd like to create a Task Group for these tasks, you can create a group named "replicate" and add them all to it. The below is just an example for how you may wish to set the `ui.taskgroups` and `ui.taskgroup.*settings`. It creates two Task Groups: (1) a "General Purpose Tasks" group for a few default DSpace Curation Tasks, and (2) a "Replication Suite Tasks" group for all these new Replication tasks.

```

# Tasks may be organized into named groups which display together in UI drop-downs
ui.taskgroups = \
  general = General Purpose Tasks, \
  replicate = Replication Suite Tasks

# Group membership is defined using comma-separated lists of task names, one property per group
ui.taskgroup.general = profileformats, requiredmetadata, checklinks
ui.taskgroup.replicate = estaipsize, readodometer, transmitaip, verifyaip, fetchaip, auditaip,
removeaip, restorefromaip, replacewithaip

```

2. **Replication Suite Configuration:** Next, in your `[dspace]/config/modules/replicate.cfg` you will want to ensure it is setup to properly use BagIt-based AIPs. Under the "AIP Packaging Settings" you'll want the following settings enabled:

```

# Package type. Permitted values: 'mets', 'bagit'
# mets = Generate default DSpace AIPs as described in: https://wiki.duraspace.org/display/DSDOC18
/AIP+Backup+and+Restore
# bagit = Generate AIPs based on the BagIt packaging format: https://wiki.ucop.edu/display/Curation/BagIt
packer.pkgtype = bagit

```

Storage Options

Where your AIPs will be stored is the next decision to make. There are three options currently available:

1. **Local Storage:** Replicate/Backup content to another location (folder) on your local filesystem.
2. **Mountable Storage:** Replicate/Backup content to a mounted external filesystem (e.g. NFS-mounted drive).
3. **DuraCloud Storage:** Replicate/Backup content to an existing [DuraCloud](#) account.

Configuring Local Storage

The local storage option may also be used for a mounted drive / SAN which just appears as though it is a local filesystem folder. However, some mounted drives (e.g. NFS-mounted drives) may need to use the [Mountable Storage](#) option instead.

Before configuring a local storage option, please ensure you have enough space available on your local hard drive (or mounted drive/SAN if your local folder is actually remote storage). You can use the "Estimate Storage Space for AIP(s)" (`estaipsize`) task to estimate the amount of new storage space you will need.

To configure local storage, please change the following settings in your `[dspace]/config/modules/replicate.cfg` configuration file:

1. **Enable Local Storage Plugin:** Ensure the Replication suite is setup to use the 'LocalObjectStore' plugin

```

# Replica store implementation class (specify one)
plugin.single.org.dspace.ctask.replicate.ObjectStore = \
  org.dspace.ctask.replicate.store.LocalObjectStore

```

2. **Configure Local Storage Folder:** Configure the location where you want all AIPs to be stored on your local filesystem. This defaults to the `[dspace]/repstore` folder. *However, we recommend changing this to at least a separate hard drive from your existing DSpace installation directory!* This ensures that all your content will not be lost in the case of a hard drive failure.

```
# Location of local (e.g. local, mountable, sync) object store
# ignored for non-local stores (e.g. DuraCloud)
store.dir = ${dspace.dir}/repstore
```

- 3. Optionally Configure Subfolder Settings:** Optionally, you can configure the sub-folder names (under `store.dir`) which will be used to store AIPs, checkm manifests (if enabled), etc.

```
# The primary storage group / folder where AIPs are stored/retrieved when AIP based tasks
# are executed (e.g. "Transmit AIP", "Restore from AIP")
group.aip.name = aip-store

# The storage group / folder where Checkm Manifests are stored/retrieved when Checkm Manifest
# based tasks are executed (org.dspace.ctask.replicate.checkm.*).
group.manifest.name = manifest-store

# The storage group / folder where deletion records are kept when an object deletion occurs
# and the ReplicationConsumer is enabled (see below). Each time an object is deleted in DSpace,
# a DELETION-RECORD@[handle] file is written to this location. The deletion record is always in
# BagIt format. It details basic info about the deleted object (along with any deleted child/member
# objects)
# This deletion record may be used to restore those deleted object(s) at a later time (using "Restore
# from AIP" tasks),
# or may be used to permanently remove their AIP(s) from storage (using "Remove AIP" task).
group.delete.name = deletions
```

Using Subfolders

Your "group.aip.name", "group.manifest.name" and "group.delete.name" settings also support subfolder paths. For example:

```
group.aip.name = dspace-backup/aip-store

group.manifest.name = dspace-backup/manifest-store

group.delete.name = dspace-backup/deletions
```

With the above settings in place, all your DSpace content will be stored in the "dspace-backup" folder (under `store.dir`). AIPs will all be stored under the subfolder "aip-store/". Manifests will all be stored under the subfolder "manifest-store/". And any object deletion records will be stored under the subfolder "deletions/".

Configuring Mountable Storage

Before configuring a mounted storage option, please ensure you have enough space available on your external, mounted drive/SAN. You can use the "Estimate Storage Space for AIP(s)" (`estaipsize`) task to estimate the amount of new storage space you will need.

To configure local storage, please change the following settings in your `[dspace]/config/modules/replicate.cfg` configuration file:

- 1. Enable Local Storage Plugin:** Ensure the Replication suite is setup to use the 'MountableObjectStore' plugin

```
# Replica store implementation class (specify one)
plugin.single.org.dspace.ctask.replicate.ObjectStore = \
    org.dspace.ctask.replicate.store.MountableObjectStore
```

- 2. Configure Mounted Folder:** Configure the location where you want all AIPs to be stored. The folder should already be mounted on your local filesystem. This defaults to the `[dspace]/repstorefolder`.

```
# Location of local (e.g. local, mountable, sync) object store
# ignored for non-local stores (e.g. DuraCloud)
store.dir = ${dspace.dir}/repstore
```

- 3. Optionally Configure Subfolder Settings:** Optionally, you can configure the sub-folder names (under `store.dir`) which will be used to store AIPs, checkm manifests (if enabled), etc.

```
# The primary storage group / folder where AIPs are stored/retrieved when AIP based tasks
# are executed (e.g. "Transmit AIP", "Restore from AIP")
group.aip.name = aip-store

# The storage group / folder where Checkm Manifests are stored/retrieved when Checkm Manifest
# based tasks are executed (org.dspace.ctask.replicate.checkm.*).
group.manifest.name = manifest-store

# The storage group / folder where deletion records are kept when an object deletion occurs
# and the ReplicationConsumer is enabled (see below). Each time an object is deleted in DSpace,
# a DELETION-RECORD@[handle] file is written to this location. The deletion record is always in
# BagIt format. It details basic info about the deleted object (along with any deleted child/member
# objects)
# This deletion record may be used to restore those deleted object(s) at a later time (using "Restore
# from AIP" tasks),
# or may be used to permanently remove their AIP(s) from storage (using "Remove AIP" task).
group.delete.name = deletions
```

Using Subfolders

Your "group.aip.name", "group.manifest.name" and "group.delete.name" settings also support subfolder paths. For example:

```
group.aip.name = dspace-backup/aip-store

group.manifest.name = dspace-backup/manifest-store

group.delete.name = dspace-backup/deletions
```

With the above settings in place, all your DSpace content will be stored in the "dspace-backup" folder (under `store.dir`). AIPs will all be stored under the subfolder "aip-store/". Manifests will all be stored under the subfolder "manifest-store/". And any object deletion records will be stored under the subfolder "deletions/".

Configuring DuraCloud Storage

How DuraCloud storage works

The Replication Task Suite includes a DuraCloud Storage plugin which utilizes the [DuraCloud REST API](#) to send/retrieve content to/from DuraCloud. This allows one to backup & restore DSpace via DuraCloud.

- Before you can use the DuraCloud Storage plugin, you first must [signup for a DuraCloud account](#) (or [signup for a trial account](#)).
- Once you have a DuraCloud account, you can configure the Replication Task Suite to use your [DuraCloud Account Settings](#) (as detailed below).
- In DuraCloud, you will also want to create one (or more) "DuraCloud Spaces" in which to store your DSpace AIPs. You'll then need to configure those space(s) in the [DuraCloud Storage Settings](#) of the Replication Task Suite (as detailed below). The DuraCloud Space represents the location in your DuraCloud account where you want DSpace to store its content. Having a separate DuraCloud Space for your DSpace content is recommended (though not required), as it allows you to separate your DSpace content from any other content you may wish to store in DuraCloud.

When you backup (transmit) DSpace content to DuraCloud via the Replication Task Suite, the following general steps occur:

1. For each DSpace object (Community, Collection, Item), an AIP zip file is generated on the server running DSpace. The AIP is temporarily stored in the server's `[dspace]/replicate/[group.aip.name]` directory, where `"[group.aip.name]"` is the value of the "group.aip.name" setting in your "replicate.cfg" configuration file (see [DuraCloud Storage Settings](#) below for more info). This "group.aip.name" setting also corresponds to the ID of the DuraCloud Space where the AIP will be stored.
2. Once the AIP is generated, the Replication Task Suite determines whether a file of this same name already exists in the DuraCloud Space.
 - a. If this file does not exist in DuraCloud, the locally generated AIP is uploaded to DuraCloud.
 - b. If a file of this name already exists, then the Replication Task Suite checks to see if it differs from the locally generated AIP. It does so by verifying the DuraCloud reported checksum with the locally generated checksum.
 - i. If the AIP checksums differ, the locally generated AIP is uploaded to DuraCloud and it replaces the version that was previously in DuraCloud.
 - ii. If the AIP checksums are identical, then the AIP is skipped. *Nothing is uploaded to DuraCloud as the files are identical.* This ensures that unnecessary uploads to DuraCloud are avoided.
3. Once the local copy of the AIP is no longer needed, it is removed from the server's temporary location.
4. If an upload to DuraCloud occurred, the local "odometer" is incremented to ensure it always details the total amount of content that has been uploaded (see [Keeping Score](#) section for more info on the "odometer").

When you restore/replace DSpace content from DuraCloud via the Replication Task Suite, the following general steps occur:

1. For each DSpace object (Community, Collection, Item), that object's AIP is downloaded from DuraCloud to the server running DSpace (the appropriate AIP is located in DuraCloud via its filename). The AIP is temporarily stored in the server's `[dspace]/replicate/[group.aip.name]` directory, where `"[group.aip.name]"` is the value of the "group.aip.name" setting in your "replicate.cfg" configuration file (see [DuraCloud Storage Settings](#) below for more info). This "group.aip.name" setting also corresponds to the ID of the DuraCloud Space where the AIP is stored.
2. Once the download completes, the local "odometer" is incremented to ensure it always details the total amount of content that has been downloaded (see [Keeping Score](#) section for more info on the "odometer").
3. The AIP is then "unzipped", and the DSpace object is restored/replaced as needed.

4. Once the local copy of the AIP is no longer needed, it is removed from the server's temporary location.

Whether you are backing up content to DuraCloud or restoring content from DuraCloud, the Replication Task Suite helps to ensure that these tasks are as seamless as possible. As moving content in/out of the cloud can sometimes result in extra costs, the Replication Task Suite also ensures it avoids unnecessary uploads. Finally, the Replication Task Suite helps you better estimate what those costs may be by keeping a running total of uploads/downloads in the "odometer".

DuraCloud Account Settings

In order to configure DuraCloud Storage, you first must have an existing [DuraCloud Account](#) (or a [trial account](#)). This account's settings should be configured in your `[dspace]/config/modules/duracloud.cfg` file as follows:

1. **DuraCloud HostName:** This is the location of your DuraCloud instance (the URL you tend to access for your account). Just provide the hostname.

```
# DuraCloud service location (just the hostname)
host = demo.duracloud.org
```

2. **DuraCloud Service Port:** This is the port that DuraCloud is running on. It is almost always "443", unless you have installed DuraCloud yourself and configured it differently.

```
# DuraCloud service port (usually 443 for https)
port = 443
```

3. **DuraCloud's "DuraStore" path:** This the path to DuraCloud's "DuraStore" service. It is almost always "durastore", unless you have installed DuraCloud yourself and configured it differently.

```
context = durastore
```

4. **DuraCloud Username & Password:** Finally, fill out your account username & password in these settings. Please note, as this file now contains your DuraCloud account information, we recommend securing it (if possible). Just ensure it is still readable by the system user that DSpace runs as.

```
# DuraCloud user name
username = myduraclouduser
# DuraCloud password
password = passw0rd
```

DuraCloud Storage Settings

Now, to configure DuraCloud as your storage location please change the following settings in your `[dspace]/config/modules/replicate.cfg` configuration file:

1. **Enable DuraCloud Storage Plugin:** Ensure the Replication suite is setup to use the 'DuraCloudObjectStore' plugin

```
# Replica store implementation class (specify one)
plugin.single.org.dspace.ctask.replicate.ObjectStore = \
    org.dspace.ctask.replicate.store.DuraCloudObjectStore
```

2. **Configure DuraCloud Primary Space to use:** Your DuraCloud account allows you to separate content into various "Spaces". You'll need to create a new DuraCloud Space that your AIPs will be stored within, and configure that as your `group.aip.name` (by default it's set to a DuraCloud Space with ID of "aip-store").


```
# The primary storage group / folder where AIPs are stored/retrieved when AIP based tasks
# are executed (e.g. "Transmit AIP", "Restore from AIP")
group.aip.name = aip-store
```

3. **Optionally, Configure Additional DuraCloud Spaces:** If you have chosen to utilize [Checksum manifest validation](#), you will need to create and configure a DuraCloud Space corresponding to the `group.manifest.name` setting below. Additionally, if you have chosen to enable the [Automatic Replication](#), you will need to create and configure a DuraCloud Space corresponding to the `group.delete.name` setting below.

```
# The storage group / folder where Checkm Manifests are stored/retrieved when Checkm Manifest
# based tasks are executed (org.dspace.ctask.replicate.checkm.*).
group.manifest.name = manifest-store

# The storage group / folder where deletion records are kept when an object deletion occurs
# and the ReplicationConsumer is enabled (see below). Each time an object is deleted in DSpace,
# a DELETION-RECORD@[handle] file is written to this location. The deletion record is always in
# BagIt format. It details basic info about the deleted object (along with any deleted child/member
# objects)
# This deletion record may be used to restore those deleted object(s) at a later time (using "Restore
# from AIP" tasks),
# or may be used to permanently remove their AIP(s) from storage (using "Remove AIP" task).
group.delete.name = deletions
```

Using File Prefixes instead of separate DuraCloud Spaces

 If you'd rather keep all your DSpace files in a **single** DuraCloud Space, you can tweak your "group.aip.name", "group.manifest.name" and "group.delete.name" settings to specify a file-prefix to use. For example:

```
group.aip.name = dspace-backup/aip-store

group.manifest.name = dspace-backup/manifest-store

group.delete.name = dspace-backup/deletions
```

With the above settings in place, all your DSpace content will be stored in the "dspace-backup" Space within DuraCloud. AIPs will all be stored with a file-prefix of "aip-store/" (e.g. "aip-store/ITEM@123456789-2.zip"). Manifests will all be stored with a file-prefix of "manifest-store/". And any object deletion records will be stored with a file-prefix of "deletions/". This allows you to keep all your content in a single DuraCloud Space while avoiding name conflicts between AIPs, Manifests and deletion records.

Automation Options (Recommended)

Performing a backup of DSpace is one thing..but ensuring that backup is always "synchronized" with your changing DSpace content is another.

The Replication Task Suite offers several options to automate replication of content to your backup storage location of choice.

1. [Automatically Sync Changes \(via Queue\)](#) : Any changes that happen in DSpace (new objects, changed objects, deleted objects) are automatically added to a "queue". This queue can then be processed on a schedule (via cron).
2. [Scheduled Site Auditing/Replication](#) : You may also wish to perform a full site audit or backup on a scheduled basis.

Automatically Sync Changes (via Queue)

The Replication Task Suite includes an 'event consumer', that can 'listen for' any changes to objects in the repository. The job of this 'consumer' is to ensure that anytime an object is added/changed/deleted, it is added to the queue of objects that need to be replicated to your backup storage location.

Activate the Sync Consumer

In order to enable/activate synchronization, you will need to add a new consumer to the list of DSpace consumers (in `dspace.cfg`). It is recommended to add this new configuration to the end of the list of existing "event.consumer." options in your `dspace.cfg` file.

- **METS-based AIP Replicate Consumer:** This consumer will listen for changes to any DSpace Communities, Collections, Items, Groups, or EPeople. It should be utilized if you have chosen to use METS-based AIPs. See [AIP Format Options](#) above for more details.

```
#### Event System Configuration ####

# ADD the "replicate" consumer to the end of the list of 'default.consumers' (This enables the consumer)
event.dispatcher.default.consumers = versioning, search, browse, discovery, eperson, harvester, replicate
....

# Configure consumer to manage METS AIP content replication
event.consumer.replicate.class = org.dspace.ctask.replicate.METSReplicateConsumer
event.consumer.replicate.filters = Community|Collection|Item|Group|EPerson+All
```

- In human terms, this configuration essentially means: listen for **all** changes to Communities, Collections, Items, Groups and EPeople. If a change is detected, run the "METSReplicateConsumer" (which adds that object to the queue).
- **BagIt-based AIP Consumer** : This consumer will ONLY listen for changes to DSpace Communities, Collections and Items as those are the only types of objects which are stored in BagIt-based AIPs. See [AIP Format Options](#) above for more details

```
#### Event System Configuration ####

# ADD the "replicate" consumer to the end of the list of 'default.consumers' (This enables the consumer)
event.dispatcher.default.consumers = versioning, search, browse, discovery, eperson, harvester, replicate

....

# Configure consumer to manage BagIt AIP content replication
event.consumer.replicate.class = org.dspace.ctask.replicate.BagItReplicateConsumer
event.consumer.replicate.filters = Community|Collection|Item+Install|Modify|Modify_Metadata|Delete
```

- In human terms, this configuration essentially means: listen for any new, modified or deleted Items, Collections and Communities. If you do not care about Community or Collection AIPs, just remove 'Community' or 'Collection' from the list. When one of the specified changes is detected, run the "BagItReplicateConsumer" (which adds that object to the queue).

You will need to restart DSpace for this new Consumer to be recognized.

How the Sync Consumer works

When the activated ReplicateConsumer detects a change on an object (Community, Collection or Item) in DSpace, it will do the following:

- **Object is created/added in DSpace:** If the event is an addition of a new DSpace object (for items this only occurs once the item exits approval workflow), then a request for an AIP transmission is queued.
- **Object is changed/modified in DSpace:** The same occurs whenever an object has changed (so-called modify events). The modified object is queued for AIP transmission.
- **Object is deleted from DSpace:** When an object is deleted, a 'record' of the deletion is transmitted to the replication service. The deletion record is stored in your configured "group.delete.name" (in replicate.cfg) and named DELETION-RECORD@[handle].zip. The deletion record is a BagIt package which simply lists all the objects that were deleted: if an item, then just the handle of the item, if a collection, then all the item handles that were in it. This way, if the deletion was a mistake, the "deletion record" can be used to recover all the contents. This represents the default behavior of the consumer. However, you may configure it in [dspace]/modules/replicate.cfg.
 - It is worth noting that when you delete an object in DSpace, the Sync Consumer will NOT delete that object's AIP from storage. All it does is write a "DELETION-RECORD@[handle]" file to your configured storage location. This ensures that you can review those "deletion records" at a later time, and decide whether to permanently delete the AIP(s) from storage, or alternatively restore the deleted object(s) in DSpace from their AIP(s).
 - *How to restore deleted objects:* Simply run one of the "Restore from AIP" commands available in the Replication Task Suite, passing it the handle of the deleted object. That command will locate the associated "DELETION-RECORD" file and appropriately restore any objects listed in that deletion record. Once the restoration is complete, the associated "DELETION-RECORD" file will be removed.
 - *How to permanently delete objects' AIPs from storage:* If an object deletion was found to be valid, you may wish to permanently remove the deleted object's AIP from remote storage (to save storage space). Simply run the "Remove AIP" command, passing it the handle of the deleted object. That command will permanently delete the object's AIP along with any associated "DELETION-RECORD" file from your storage location. (**WARNING: once an AIP is deleted, you will be unable to restore that object to DSpace in the future.**)

Configuring the Sync Consumer

The actions of the activated ReplicateConsumer (i.e. both METSReplicateConsumer and BagItReplicateConsumer) is configured within the [dspace]/modules/replicate.cfg. Below are the default options (which normally need no modification)

```

### ReplicateConsumer settings ###
# ReplicateConsumer must be properly declared/configured in dspace.cfg
# All tasks defined will be queued, unless the '+p' suffix is appended, when
# they will be immediately performed. Exercise considerable caution when using
# +p, as lengthy tasks can adversely affect UI or other responsiveness.

# Replicate event consumer tasks upon install/add events.
# A comma separated list of valid task plugin names (with optional '+p' suffix)
# By default we transmit a new AIP when a new object is added
consumer.tasks.add = transmitsingleaip

# Replicate event consumer tasks upon modification events.
# A comma separated list of valid task plugin names (with optional '+p' suffix)
# By default we transmit an updated AIP when an object is modified
consumer.tasks.mod = transmitsingleaip

# Replicate event consumer tasks upon a delete/remove events.
# A comma separated list of valid task plugin names (with optional '+p' suffix)
# By default we write out a deletion catalog & move the deleted object's AIP
# to the "trash" group in storage (where it can be permanently deleted later)
consumer.tasks.del = catalog+p


# Replicate event consumer queue name - where all queued tasks are placed
# This queue appears under the curate.cfg file's 'taskqueue.dir'
# (default taskqueue location is [dspace]/ctqueues/)
consumer.queue = replication

```

As you can see in the default configuration above...

- Both "add" and "modification" events add the "transmitsingleaip" task (which will regenerate & transmit the object AIP to replica storage) to the queue of tasks to perform. Please ensure you are scheduling this queue to be processed, as detailed in [Processing the Consumer Queue](#) below.
- The "delete" event triggers a special "catalog" task. This "catalog" task does the following:
 - First, it creates a plaintext "catalog" file which lists all the objects that were deleted.
 - Second, it moves the AIPs for those deleted objects to the "group.delete.name" storage area (this is essentially putting them in a "trash" folder, where they can be cleaned up later, or potentially restored if the deletion was accidental).
- By default, the queue used for all replication events is located at: [dspace]/ctqueues/replication (this is a plaintext file which just lists all actions that should be performed the next time the queue is processed)

Including or Excluding specific objects from automatic sync

 It is also possible to include or exclude specific objects (via their handles) from this automatic sync. This can be done via the addition of an "include" or "exclude" textfile in your "base.dir" (by default: [dspace]/replicate/).

For example, suppose you only want to synchronize a single important Community (with handle "123456789/10"), you can create a textfile named "include" and add a single line of text:

- 123456789/10


Alternatively, if you'd like to synchronize everything **except** for two unimportant Collections (with handles: "123456789/11" and "123456789/12"), you can create a text file named "exclude" and add two lines of text:

- 123456789/11
- 123456789/12

In either the "include" or "exclude" files, you can add as many handles (one per line) as you like. These handles can represent Communities, Collections or Items.

Please note that the "exclude" file takes precedence over the "include" file. So, if an object handle is listed in both files, that object will be excluded from processing.

Don't forget to schedule the Consumer Queue to be processed!

 By default, just configuring the Consumer will only generate a queue of tasks in the location specified by the `consumer.queue` setting in "replicate.cfg". You must ensure that you schedule this queue to be processed for the synchronization to be complete. See the [Processing the Consumer Queue](#) section below.

Processing the Sync Consumer Queue

Once you've setup your Consumer & restarted DSpace, you'll start to see a (plain text file) queue of tasks (in the location specified by the `consumer.queue` setting in "replicate.cfg") that need to be performed in order to synchronize your AIP backup with what is in your DSpace instance. This replication queue is just a normal DSpace [Curation System](#) queue, and it can be processed via command line or a cron job (recommended).

Processing this queue is as simple as scheduling a cron job (or similar) to run on a daily, weekly or monthly basis (how frequently you want this to run is up to you). For example, here's a cron job which will process a queue named "replication" every Saturday at 1:00AM local time and places the results into a "replication-queue.log" file (NOTE: you may need to modify the paths to DSpace below if you wish to use this example):


```
0 1 * * 6 $HOME/dspace/bin/dspace curate -q replication -r - > $HOME/dspace/log/replication-queue.log 2>&1
```

- The "-r -" part of this command ensures that the results of the sync are reported back to you on the command line, rather than being logged to the main dspace.log file.
- Then the "> replication-queue.log 2>&1" takes those reported results (success & errors) and writes them to a "replication-queue.log" file.
- Although this example is a weekly sync, some institutions may wish to perform a more frequent (daily) or less frequent (monthly) sync based on local policies (especially based on whether AIPs are the sole form of backup, or acting as more of a "secondary" safety backup)

In case it is not obvious, you can also process this queue manually via the command line by simply running: `[dspace]/bin/dspace curate -q replication`

During the processing of the queue, the existing queue file is "locked", but any new changes are logged to a separate queue file. In other words, changes that happen in the DSpace system during the processing of the queue will still be captured in a separate queue file (which will be processed the next time you execute the above script).

You still may wish to perform an occasional full site audit/backup

 Even if you are processing the "sync queue" on a daily or weekly basis, you still may want to perform a full site-wide audit and/or backup on a less frequent basis. For example, if you are processing the sync queue on a daily basis, you might want to perform a weekly or monthly site audit/backup. Although this full site audit/backup is not required, it helps to ensure that all of your AIPs are *simultaneously* update-to-date at a given point in time. It's worth noting that only AIPs that have **changed** (i.e. have a different checksum) will be transferred to your backup location. So, if all AIPs are already up-to-date in your backup location, no AIPs would even be transferred.

More information on performing such an "audit" or full-site backup (including cron job examples) can be found in the section on [Scheduled Site Auditing / Replication](#)

Enhancing the Performance of the Queue Processing (optional)

For large or highly active repositories, the "replication" queue may grow rather large as each change in the system adds (at least) one task to the queue. In addition, if the same object is modified multiple times (e.g. several small tweaks to an item), it will cause duplicate entries to appear in this queue.

In DSpace, by default, duplicate tasks in a [Curation System](#) queue will each be processed individually. So, that means if an Item is updated 10 times, it will appear in the queue 10 times, and its AIP will be (re-)generated and (re-)transmitted to storage 10 times when that queue is processed. (*DuraCloud Note:* Some storage platforms, e.g. DuraCloud, provide a way to determine whether a newly generated AIP actually differs from the one in replica storage. So, in the case of DuraCloud storage, the AIP will be re-generated 10 times, but it will only be transmitted to DuraCloud ONCE. The other 9 times, the DuraCloud storage plugin will determine that the checksum of the new AIP is identical to the one in DuraCloud and skip the transmission step. See [How DuraCloud storage works](#) section above for more info.)

To help resolve this issue of potentially running many duplicate tasks when you process the replication queue, the Replication Task Suite provides a specialized "FilteredFileTaskQueue" (`org.dspace.ctask.replicate.FilteredFileTaskQueue`) which can be enabled. The "FilteredFileTaskQueue", acts similar to the default "FileTaskQueue" (used by the DSpace Curation System), but it first filters out any known duplicate entries (lines) before the queue is processed. A duplicate entry is one that performs the exact same task(s) on the exact same object.

For example, given a queue file that looks like (note, the format of each entry in the queue is: "[username]|[timestamp]|[tasks]|[obj-handle]")

```
user1@myu.edu|123456789|transmitaip|10673/0
user2@myu.edu|123456790|transmitaip|10673/1
user2@myu.edu|123456791|transmitaip|10673/0
user1@myu.edu|123456792|transmitaip|10673/0
```

The default "FileTaskQueue" will execute all four entries in this queue, whereas the "FilteredFileTaskQueue" will only execute the **first two entries** (as entries #3 and #4 would be considered duplicates of entry #1).

To enable the FilteredFileTaskQueue, you would need to change the queue class specified in the `[dspace]/config/modules/curate.cfg` file and restart DSpace:

```
## task queue implementation
#plugin.single.org.dspace.curate.TaskQueue = org.dspace.curate.FileTaskQueue
plugin.single.org.dspace.curate.TaskQueue = org.dspace.ctask.replicate.FilteredFileTaskQueue
```

Please note: this change to curate.cfg will cause the entire DSpace Curation System (not just the Replication Task Suite tasks) to utilize the FilteredFileTaskQueue for queuing. This should not cause any issues with other tasks, as the FilteredFileTaskQueue is an extension of FileTaskQueue, but it's worth noting that this change will effect all curation tasks.

Scheduled Site Auditing/Replication

Whether you decide to automatically synchronize your replica (backup) store or not, you may also wish to schedule some occasional auditing or even a full "refresh" of your backup. Note that you need not necessarily perform **both** tasks, it's really up to you. Refreshing your backup (transmitaip) also performs its own internal "audit" by ensuring that only AIPs which have changed will be transmitted to your backup location (thus potentially cutting down on I/O costs for hosted backup locations).

- **Auditing for site differences:** Running an audit will check if there are differences between DSpace Content and AIP backup content. A full site AIP audit can be run from the command line, or scheduled via a cron job (or similar). For example, the following command will run a site-wide audit for a DSpace site with a handle.prefix of "10673" (and writes the results of the audit to a "siteaudit.log" file).

```
[dspace]/bin/dspace curate -t auditaip -i 10673/0 -r - > siteaudit.log 2>&1
```

- This command runs the "auditaip" task on your entire site (The identifier "[handle-prefix]/0" refers to the entire DSpace Site...in this case we used the example handle-prefix of "10673")
- The "-r -" part of this command ensures that the results of the audit are reported back to you on the command line, rather than being logged to the main dspace.log file.
- Then the "> siteaudit.log" takes those reported results and writes them to a "siteaudit.log" file.
- **A refresh of your full backup:** Whether or not you decide to use the sync queue for automated backups (as described above), you may want to ensure that you are re-running your entire site backup on a scheduled basis. For example, the following command will regenerate & transmit AIPs for every object in a DSpace site with a handle.prefix of "10673" (and writes the results of the audit to a "sitebackup.log" file). It's nearly identical to the "auditaip" command above, except that you are executing "transmitaip" and writing to a different log file. *Please note: only AIPs which have changed (i.e. have a different checksum) will be transferred to your backup location. So if you are also running automated syncing, this command may just act as way to double check that all your AIPs are up-to-date in your backup location.*

```
[dspace]/bin/dspace curate -t transmitaip -i 10673/0 -r - > sitebackup.log 2>&1
```

Additional Options

Configuring usage of Checkm manifest validation

This section goes through the steps of configuring the usage of Checkm manifest tasks. These tasks provide a capability to store DSpace content checksums external from DSpace in the [Checkm Manifest format](#). Some institutions may find this to be a useful replacement for the default [DSpace Checksum Checker/Validator](#), which only stores/validates checksums internal to the DSpace system.

However, as this is an optional set of tasks, they are disabled by default. Should you wish to enable these tasks, just do the following:

1. **General Curation Configuration:** First, in your [dspace]/config/modules/curate.cfg you will want to enable & configure the Checkm Manifest tasks. (NOTE: there is a sample curate.cfg file provided in <https://github.com/DSpace/dspace-replicate/tree/master/config/modules> which provides example settings, though they are all commented out by default).
 - **Enable the Checkm Tasks:** In the list of "Task Class implementations" (plugin.named.org.dspace.curate.CurationTask), add the following.
REMEMBER to add a comma and backslash (", \") after each line (except the final line).

```
plugin.named.org.dspace.curate.CurationTask = \  
... (YOUR EXISTING TASKS) ... , \  
org.dspace.ctask.replicate.checkm.TransmitManifest = transmitmanifest, \  
org.dspace.ctask.replicate.checkm.VerifyManifest = verifymanifest, \  
org.dspace.ctask.replicate.checkm.FetchManifest = fetchmanifest, \  
org.dspace.ctask.replicate.checkm.CompareWithManifest = auditmanifest, \  
org.dspace.ctask.replicate.checkm.RemoveManifest = removemanifest
```

- **Give Each Task a Human-Friendly Task Name:** Under the ui.tasknames setting, give each of the above Tasks a human-friendly name. Here are some recommended values, but you are welcome to tweak them.
REMEMBER to add a comma and backslash (", \") after each line (except the final line).

```
ui.tasknames = \  
... (YOUR EXISTING TASK NAMES) ... , \  
transmitmanifest = Transmit Checkm Manifest to Storage, \  
verifymanifest = Verify Checkm Manifest exists in Storage, \  
fetchmanifest = Fetch Checkm Manifest from Storage, \  
auditmanifest = Audit against Checkm Manifest, \  
removemanifest = Remove Checkm Manifest from Storage
```

- **Optionally Create a Task Group:** Finally, if you'd like to create a Task Group for these tasks, you can create a group named "checkm" and add them all to it. The below is just an example for how you may wish to set the ui.taskgroups and ui.taskgroup.* settings. It creates two Task Groups: (1) a "General Purpose Tasks" group for a few default DSpace Curation Tasks, and (2) a "Checkm Validation Tasks" group for all these new Replication tasks.

```
# Tasks may be organized into named groups which display together in UI drop-downs
ui.taskgroups = \
    general = General Purpose Tasks, \
    replicate = Replication Suite Tasks, \
    checkm = Checkm Validation Tasks

# Group membership is defined using comma-separated lists of task names, one property per group
ui.taskgroup.general = profileformats, requiredmetadata, checklinks
...
ui.taskgroup.checkm = transmitmanifest, verifymanifest, fetchmanifest, auditmanifest,
removemanifest
```

Problem Statement and Usage Examples

We can suppose our data curator has identified a collection of items in her DSpace repository consisting of high-value, born-digital, and unique /irreplaceable (not held elsewhere) content (called the 'Amazing Images' collection). She prudently wishes to insure against catastrophic local loss of this content by keeping a copy or **replica** of this collection elsewhere (e.g. either on a backup drive, or even in the cloud via a service like [DuraCloud](#)). She'd prefer to replicate all her DSpace content, but realizes that storage costs over long periods has made her administration wary, so decides to begin with this collection.

First Steps - Estimation

Replication Task Used:	Estimate Storage Space for AIP(s)	Task ID: estaipsize
-------------------------------	-----------------------------------	---------------------

In order to budget for replication storage, she needs to know the 'size' of the collection. When she asks her sysadmin, he replies that it is easy to give her figures for the whole DSpace asset store, but since collections aren't stored separately, she would have to add up each item's bitstreams in the collection, a rather tedious process. Thus the first task: a reporting tool which operates on natural DSpace objects, rather than storage volumes. The "Estimate Storage Space for AIP(s)" (estaipsize) task will give her this ability.

As this is the first task we are introducing in the Replication Task suite, let's take quick look at how each task is configured/enabled. Each enabled task in this Suite is defined in `[dspace]/config/modules/curate.cfg`. So, this estaipsize task has its own definition in the file as follows:

```
plugin.named.org.dspace.curate.CurationTask = \
... other curation tasks
    org.dspace.ctask.replicate.EstimateAIPSize = estaipsize
```

In that same file, each task is given a human-readable "name", which is what is displayed in the DSpace Administrative UI:

```
ui.tasknames = \
... other tasks
    estaipsize = Estimate Storage Space for AIP(s)
```

Of course, both the name of the task ('estaipsize'), and the human-readable name can be easily modifiable in this file. You are free to rename them as you see fit.

Now, getting back to our curator. To utilize this task (or any other task in the Replication Task Suite), she logs in to DSpace, navigates to her collection, and chooses the "Edit Collection" option in her administrative interface. From that edit screen, a 'curate' tab provides her with a dropdown list of various curation tasks. Among those tasks, she will find the "Estimate Storage Space for AIP(s)" task listed, which she selects and clicks "Perform". On the page, the results will display:

ID: 123456789/1 (Amazing Images) estimated AIP size: 4 gigabytes

We should warn that the estimates from this task are rather crude, in that they do not measure the actual size of all AIPs. Rather they just total up the bitstream (file) sizes (and do not include metadata files). However, even this crude estimate should provide a decent idea of overall storage needs.

Replicating

Replication Task Used:	Transmit AIP(s) to Storage	Task ID: transmitaip
-------------------------------	----------------------------	----------------------

Having secured approval to replicate 'Amazing Images' collection, our curator obviously needs a task to generate the AIP representations of each item in the collection, and transmit these archive files to the replication storage site (which may be service-backed, local, in the cloud, etc, as will be explored below). This task is the "Transmit AIP(s) to Storage" (transmitaip) task.

Since we are now working with AIPs, we should examine how they are configured to the tasks. Most configuration specific to the replication task suite is found at `[dspace]/config/modules/replicate.cfg`. There are two main properties to set (or accept default values):

```
# Package type. Permitted values: 'mets', 'bagit'
packer.pkgtype = mets
# Format of package compression. Permitted values: 'zip' or 'tgz'
# for 'mets' packages, only zip is supported
packer.archfmt = zip
```

The default values will create a METS-based AIP in the default [DSpace AIP Format](#), compressed into a 'zip' archive. The other alternative supported by the replication task suite is Library of Congress [Bagit packaging](#), which may be compressed either into a 'zip' file or a 'tgz' ('gzipped tar'), a compression standard more common in Unix systems.

Our data curator may elect to perform this task in the DSpace Admin UI, or, if the collection is rather large, she may instead 'queue' the task for later execution by using the queuing facility available in the curation system. We should note that the 'transmitaip' task, like all other replication tasks, operates on whatever DSpace object(s) they are given. Thus, if the object is a collection, the task creates (and transmits, of course) an AIP for the collection object itself (metadata and logo), as well as AIPs for each item in the collection. If the task is given an identifier for a single Item, then only one AIP will be created and transmitted.

Verifying Replication

Replication Task Used:	Verify AIP(s) exist in Storage	Task ID: <code>verifyaip</code>
-------------------------------	--------------------------------	---------------------------------

While the 'transmitaip' task will report on whether or not it was successful in generating and transmitting AIP(s) to the replication service, our data curator wants the ability (within DSpace) to check whenever she likes that the AIP(s) which were transmitted are still there. A simple task "Verify AIP(s) exist in Storage" (`verifyaip`) can perform this function.

Ensuring Replica Integrity and Accuracy over time

Replication Task Used:	Audit against AIP(s)	Task ID: <code>auditaip</code>
-------------------------------	----------------------	--------------------------------

The 'Amazing Images' collection is comparatively static, meaning that few new items are likely to be added, and most of the metadata in each item is not routinely changed. However, over longer periods of time, cataloging errors are discovered and corrected, perhaps formats become obsolete and new bitstreams are added. If the curator is fastidious about each change, and performs the 'transmitaip' task on each item that has changed, then in general the set of AIP replicas will always be 'in sync' with the repository. However, it is useful to have the means to ensure that the replicas agree with the repository without having to create and transmit entirely new ones. Thus the task: "Audit against AIP(s)" (`auditaip`), which can also be thought of as a simple, quick auditing task. When performed on an Item, the task does the following:

1. generates an AIP for the DSpace object locally (but does not transmit it)
2. computes an MD5 checksum on the local AIP
3. requests from the replication storage service an MD5 checksum for the AIP in storage
4. compares the 2 checksums

The task will thus fail only if the checksums differ, which can only happen if some part of the DSpace Object (metadata or bitstream) itself differs. If the version of the item that is believed to be authentic is the repository (local) one, then a simple performance of 'transmitaip' task on the item will restore synchrony. For collections and communities, this task also does an 'extent' comparison, which means that it will determine whether the replica store has an AIP for every item known (locally) to be in the collection or community.

Repairing Damage

The AIPs in the replica store represent an insurance policy, and when 'claims' against that policy are filed, they can cover two situations:

- either the repository object is completely missing, and we want to restore it,
- or it is damaged and we want to repair the damage with data from the replica store AIP.

A set of replication tasks perform these functions, as described below.

Restoring Object(s)

Replication Tasks Used:	Restore Missing Object(s) from AIP(s)	Task ID: <code>restorefromaip</code>
	Restore Missing Object(s) but Keep Existing Objects (<i>*METS-AIP Only</i>)	Task ID: <code>restorekeepexisting</code>
	Restore Single Object from AIP (<i>*METS-AIP Only</i>)	Task ID: <code>restoresinglefromaip</code>

If the curator should ever find the need to restore a deleted object, a variety of restoration based tasks are available. The base task is the "Restore Missing Object(s) from AIP(s)" (`restorefromaip`) task.

This "Restore Missing Object(s) from AIP(s)" (`restorefromaip`) task will do the following:

1. fetch the replica store AIP for the given object identifier
2. decompress it and create a new DSpace object
3. install the object into the repository, including restoring its state (withdrawn, embargoed, etc.)
4. if the object is a collection or community, all child objects (e.g. items) will also have their AIP fetched, decompressed and restored

NOTE: This `restorefromaip` task will fail if there is already an object in the repository bearing the identifier given. In other words, it will report a failure if an object is found to already exist.

When utilizing METS-based AIPs, two additional restoration tasks are available:

- Restore Single Object from AIP (`restoresinglefromaip`)
 - This task acts the same as the default "restorefromaip" task, but it does NOT restore any child objects. So, if it is run on a collection, just the collection itself will be restored (items in that collection will not be restored).
- Restore Missing Object(s) but Keep Existing Objects (`restorekeepexisting`)
 - This task acts similar to the default "restorefromaip" task, but it attempts to skip over any objects which already exist in the repository. In other words, an error is not thrown if an object already exists – rather that entire object (and all its child objects) are skipped over during processing and left **unchanged**. This mode is identical to the "Keep Existing" mode of the DSpace [AIP Backup and Restore](#) tool.

Replacing Object(s)

Replication Tasks Used:	Replace Existing Object(s) with AIP(s)	Task ID: <code>replacewithaip</code>
	Replace Single Object with AIP (*METS-AIP Only)	Task ID: <code>replacesinglewithaip</code>

If the curator should ever find a need to replace a corrupted object or revert an existing object back to the version in remote storage, a variety of replacement tasks are available. The base task is the "Replace Existing Object(s) with AIP(s)" (`replacewithaip`) task.

The "Replace Existing Object(s) with AIP(s)" (`replacewithaip`) task expects to replace an existing DSpace object. This task will do the following:

1. fetch the replica store AIP for the given DSpace Object
2. decompress it
3. locate the existing DSpace object to be replaced & clear out all its existing metadata, files, access rights, etc.
4. replace the existing DSpace object metadata, files, access rights, etc. with the information found in the AIP (thus "overwriting" or replacing all information in the existing object)
5. if the object is a collection or community, all child objects (e.g. items) will also have their AIP fetched, decompressed and existing objects replaced

NOTE: When using BagIt-based AIPs, this task will fail if the DSpace object is not found or no longer exists. When using METS-based AIPs, this task will instead perform a restoration of any DSpace object that is not found or no longer exists.

If you are using METS-based AIPs, an addition replacement task is available:

- Replace Single Object from AIP (`replacesinglewithaip`)
 - This task acts the same as the default "replacewithaip" task, but it does NOT replace any child objects. So, if it is run on a collection, just the collection **metadata** will be replaced (items existing in that collection will not be replaced).

Cleanup

Replication Task Used:	Remove AIP(s) from Storage	Task ID: <code>removeaip</code>
-------------------------------	----------------------------	---------------------------------

Ordinarily, a replication arrangement is long standing: the preservation function cannot be fulfilled unless the replicas (here, the AIPs) are always kept and available. However, some collections (or items within them) may be removed for a variety of reasons: legal challenge, de-accession, etc. When the repository no longer locally wants to hold the object, the replica AIP ceases to have value. The task 'Remove AIP(s) from Storage' (`removeaip`) will **permanently** delete the replica store AIP for its identifier. As will other replication tasks, if the identifier points to collection or community, all the AIPs of all the members will also be **permanently** deleted.

Keeping Score


Replication Task Used:	Read Odometer	Task ID: <code>readodometer</code>
-------------------------------	---------------	------------------------------------

Many storage providers have cost structures that are more complex than simple functions of the total stored bytes: particularly cloud providers have costs associated with the use of the network to upload and download the stored object. An object that occupies 2 megaBytes might cost far more over time than a 1 gigaByte object, if the former is downloaded 1000 times for every time the latter is. The replication system provides a very rudimentary task to help manage and track these factors: 'Read Odometer' (`readodometer`). This task simply displays the readings from the replication system that records cumulative use. The statistics are:

- total number of objects (AIPs, typically) in the replica store
- total size of all objects
- total number of bytes downloaded from the store
- total number of bytes uploaded to the store

These figures can be used as a means of checking and validating service charges from storage providers.

More Information on where Odometer statistics are kept

 The odometer statistics are stored in a small text file located at: `[base.dir]/odometer`, where `[base.dir]` is the value of the `base.dir` setting in your `[dspace]/config/modules/replicate.cfg` configuration file. Should you ever need to reset your odometer, you can do so by moving or removing this existing odometer file.

Automation (Recommended)

While the coordinated use of the tasks described above can provide the basis for a solid replication strategy and practice, there are several processes that could necessitate a fair amount of curatorial work. For example, in the discussion on ensuring integrity of AIPs over time, we remarked that vigilance was required by the curator to transmit new AIPs whenever Items change. It is possible to leverage existing facilities in DSpace to substantially reduce this effort through automation.

The Replication Task Suite includes a so-called 'event consumer', that can 'listen for' any changes to objects in the repository. When this consumer detects that a change has occurred (e.g. object added/changed/deleted), it will automatically queue specific AIPs to be regenerated.

Using the event consumer, the curator can essentially operate replication in 'auto-pilot' after the first complete transmission of AIPs. This provides a "set it and forget it" option for your backup solution.

More information about setting up automation is available in the [Automation Options](#) configuration section above.

Replica Storage / Backup Location

For the replication of AIPs to be of any significant value, they must be stored in a safe, persistent, reliable, accessible, and available location. The replication tasks of transmitting, fetching, etc all rely on the storage provider configured.

The Replication Task Suite provides three storage provider options:

- **System Folder:** The default configuration (`LocalObjectStore`) simply writes the AIPs to the local directory configured by the 'store.dir' property in `replicate.cfg`. This is not intended to be a production-grade solution, since a failure in the DSpace asset store could likely also affect this storage (unless it is on a separate physical drive). This base option is provided mostly as a way to begin to work with the replication tasks without worrying about finding a storage provider.
- **DuraCloud:** For replicating in earnest, a service like [DuraCloud](#) is recommended (`DuraCloudObjectStore`). Such a service has the additional benefits of providing offsite storage/replication while also providing additional preservation management tools. Note that this service must be established and provisioned prior to use. For more information on DuraCloud see: <http://www.duracloud.org>
- **Mounted Drive:** Alternatively, a `MountableObjectStore` option may be used if you wish to keep your AIP storage more "local" (e.g. on a local SAN or storage network). This option acts similar to the default configuration (in that it writes to the local directory configured by the 'store.dir' property in `replicate.cfg`). But, the expectation is that directory is actually a mounted storage drive, so AIPs are written in such a way as to support more complex storage architectures (e.g. an NFS-mounted store).

More information about each of these storage options (and how to configure them) is available in the [Storage Options](#) configuration section above.

Codebase / Development

The following are notes for developers on how to checkout the Replication Task Suite code & build it from source.

1. Download the Replication Suite code from GitHub: <https://github.com/DSpace/dspace-replicate>
 - a. Checkout the branch you wish to develop against. For example, to checkout the 1.x branch of the codebase:

```
git checkout dspace-replicate-1.x
```

2. Build/Compile the Replication Suite, by running the following from the root directory

```
mvn package
```

3. The code will be compiled into a JAR and all its dependencies will also be copied to your "target" directory
 - a. The main `dspace-replicate.jar` will be compiled to:
 - `[dspace-replicate]/target/dspace-replicate-[version].jar` (The Replication Suite Plugin)
 - b. There will also be a total of 4 dependency JARs that will be copied to:
 - `[dspace-replicate]/target/lib/common-[version].jar` (DuraCloud common libraries - required for DuraCloud integration)
 - `[dspace-replicate]/target/lib/commons-compress-[version].jar` (Apache Commons Compress - prerequisite for Replication Suite plugin)
 - `[dspace-replicate]/target/lib/storageprovider-[version].jar` (DuraCloud storage provider libraries - required for DuraCloud integration)
 - `[dspace-replicate]/target/lib/storeclient-[version].jar` (DuraCloud store client libraries - required for DuraCloud integration)
4. Once the codebase is compiled, you can install it by following the [Installation](#) instructions above.
 - a. Alternatively, you may temporarily copy all 5 JARs (`dspace-replicate` + dependency JARs) to the following locations for testing purposes only:
 - DSpace "lib" folder (e.g. `[dspace]/lib/`) - This will make the Replication Task Suite available via the commandline
 - DSpace XMLUI "lib" folder (e.g. `[dspace]/webapps/xmlui/WEB-INF/lib/`) - This will make the Replication Task Suite available via the XMLUI.

- b. You will also need to follow the [Configuration](#) instructions above in order to properly enable & configure the Replication Task Suite.