Purpose

Provide an explanation on how the new RUcore File Packaging service is implemented.

Package Requests

A new method called PACKAGE has been added to the RUcore Get API. The PACKAGE method supports the POST method for interaction. The following are parameters the POST method listens for.

Parameter	Values	Description
token	string (required)	A valid token that was provided by the getToken() method
files	string (optional)	Supports a string of multiple Fedora Object ID's that are semi-colon delimited. The Object ID can be further delimited using a forward-slash followed by a specific datastream ID. Requests are case-insensitive.
map	string (optional)	A string containing the object ID and the datastream ID of the structural map to use. The object ID and the datastream ID are delimited by a forward slash.

Sample Request – Files Only

```
$_POST['files'] = "rutgers-lib:1234/mods;rutgers-lib:5678/pdf-1;rutgers-lib:5678/jpeg-1;rutgers-lib:1234"
```

In this sample request two unique object ID's are supplied, rutgers-lib:1234 and rutgers-lib:5678.

rutgers-lib:1234/mods - In the first entry for rutgers-lib:1234 a sub-delimiter is used to state only the MODS datastream is being requested.

rutgers-lib:5678/pdf-1 - For the second entry a request is being made for rutgers-lib:5678's PDF-1 datastream.

rutgers-lib:5678/jpeg-1 - The third entry is requesting rutgers-lib:5678's JPEG-1 datastream.

rutgers-lib:1234 - The fourth entry requests that all available public accessible datastreams for rutgers-lib:1234 be packaged and sent. Since the first entry is already requesting the MODS datastream, it will be removed from the list to process as it is covered by this blanket request. Duplicate files will not be packaged.

For each of the supplied object ID's a lookup will be performed for presentation structural maps. When found the presentation structure map will be used to define the packages file hierarchy. If no presentation structural map is found, then the files will be stored in a flat directory structure.

Sample Request – Structural Map Only

```
$_POST['map'] = "rutgers-lib:1234/SMAP-PRES"
```

In this sample request only structural map information is provided. The packaging service will acquire the structural map using the object ID and datastream ID. The package will then be created using the information in the structural map and the package will deliver all accessible datastreams that are defined in the structural map.

Security Token

A new method called getToken() has been added to the RUcore Get API. This method will generate a valid token that can be used when requesting a package. The following are the required POST parameters that must be presented to generate a token.

Parameter	Values	Description
key	string (required)	A unique key passed by the application that is rendering the user interface for downloading a package.

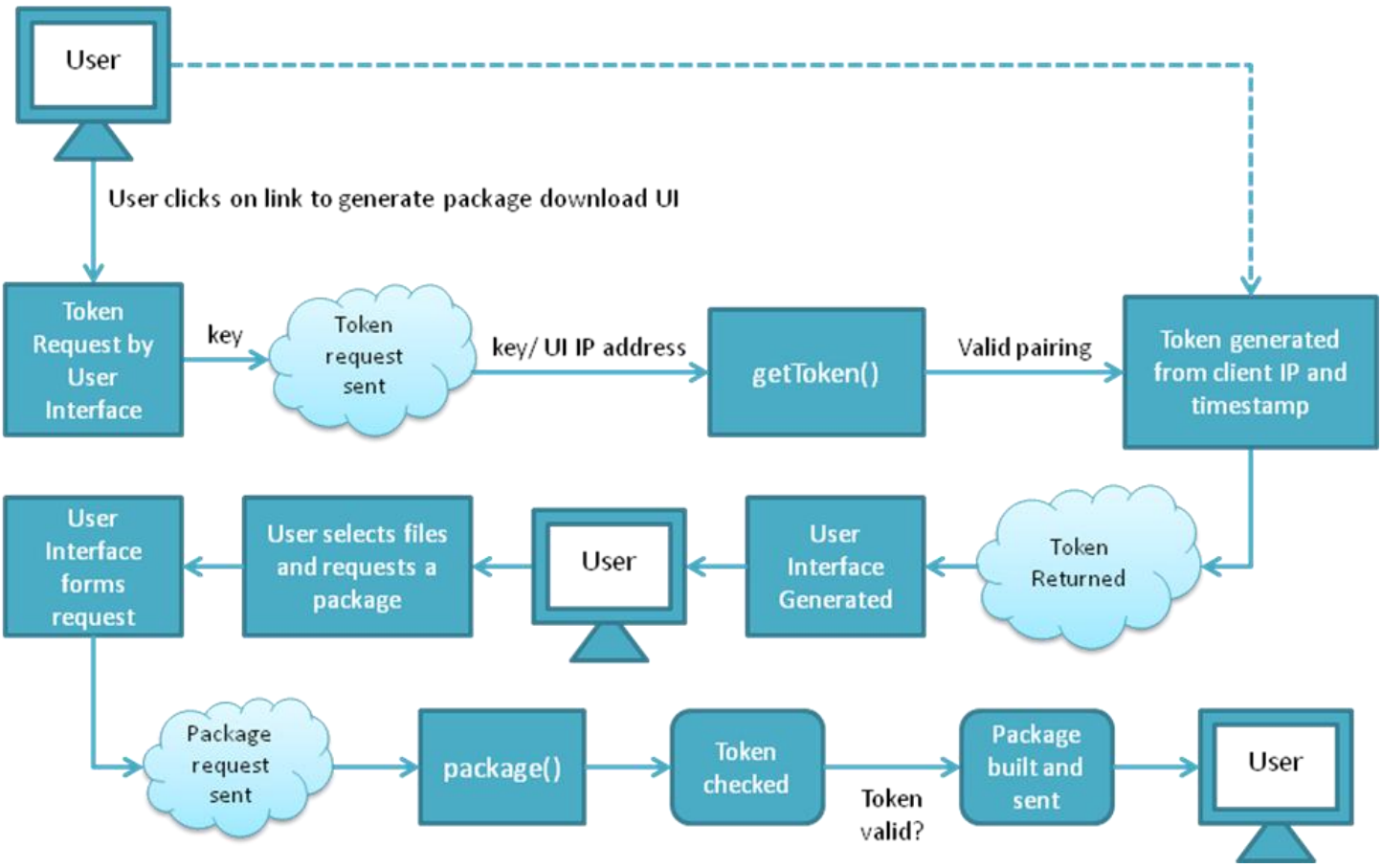
Once a key is submitted to the getToken() method the key will be paired with the IP address of the application rendering the user interface. If this pairing is valid, then a token will be generated that will be valid for a set period of time on the clients machine. This period of time will be configurable and by default will be 30 minutes.

Directly accessing the PACKAGE method without having a token generated will yield a HTTP status code 403 Forbidden.

All responses will be XML. Below are the two types of responses, <token> or <error>, which will be returned.

```
<token>MTI4LjYuMTk4Ljk5IyMyMjcxMzYuMTk4MDQwNzc=</token>
```

```
<error>Token could not be generated</error>
```



Datastream Validation

All datastream requests are vetted and validated as they normally are using the Get API's datastream retrieval functionality.

If a request is made for all publically accessible datastreams for a certain object only datastreams we have configured to be publically accessible will be packaged and sent. No ARCH, POLICY, RELS-INT or other datastreams we have reserved for internal use, archival purposes, or other reasons will be packaged. Any direct requests for those datastreams will also be ignored and dropped.

If a request is made for an object that doesn't exist, then an empty package will be returned.

If a request is made for a datastream that does not exist, then that datastream will be dropped during the datastream ID validation portion of the process.

If a request is made for a datastream that is embargoed or for a whole object that has some embargoed datastreams those datastreams will not be returned.

A file will be generated to store and report any validation issues. Following the Bag It schema the file will be an additional tag file that will be stored in the top-level of the package. The file name will be notices.txt and a tag manifest file will also be generated contained the checksum information for the notices.txt tag file.

Checksums

Part of the Bag It packaging specification requires a manifest file with checksum information for all parts of the package.

For datastreams that are managed, i.e. a PDF, if a checksum appears in the `contentDigest` portion of the object XML that value will be used in the Bag It manifest. If no checksum can be found then one will be calculated on the fly using the checksum algorithm configured in the Get API's configuration file, `$cfg_api_get['package_checksum_type'].`

For datastreams that are in-line, i.e. MODS, no checksum information appears in the object XML. The inline datastream will be loaded into a string and have its checksum calculated on the fly using the checksum algorithm configured in the Get API's configuration file, `$cfg_api_get['package_checksum_type'].`

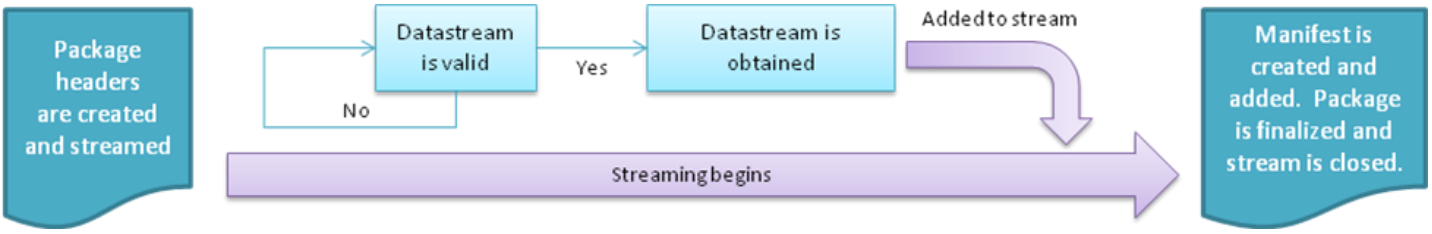
Currently the checksum algorithm is configured to be sha256 to align itself with the coming changes to the RUcore Fedora checksum policy.

Limiting Package Size

In theory the number of files in the package and the total size of the package could become quite large. It is advisable we discuss and define what "large" is and some hard limits be set. Two options we can explore for limiting the size of the package are either by the total number of files or total accumulated size of all files being requested. By default the total number of files will be set to 1000 and the total accumulated size limit will be 100 Gigabytes.

Creating the Package Using Compression

As requested objects are being analyzed and their datastreams are validated the datastreams are streamed back as a single file download. The package is being built dynamically in pieces and sent when each piece is ready. Once a datastream is sent the next datastream that has been validated is sent. Once all datastream are sent the Bag It manifest file is created in memory and sent. Finally the package finalized and the transfer is completed.

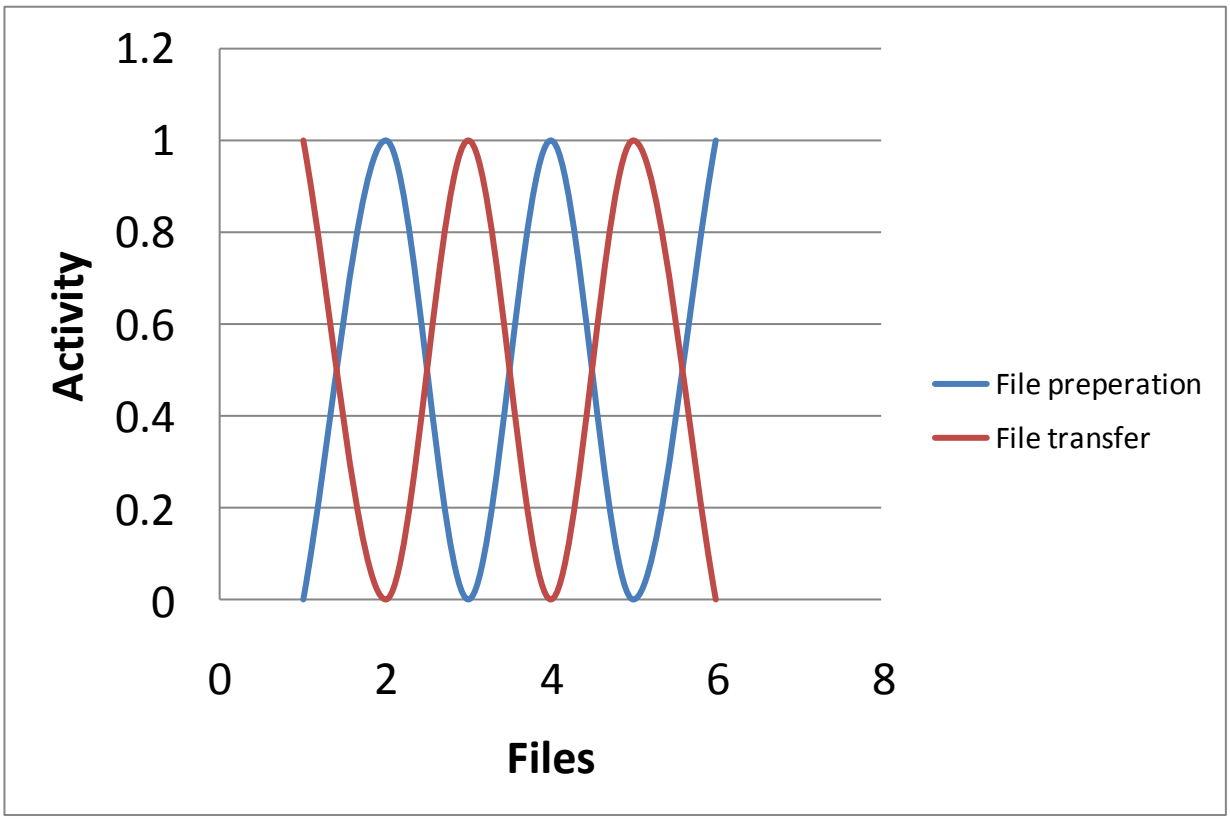


Inline datastreams, i.e. MODS, and managed datastreams, i.e. PDF's etc, are prepared differently. Inline datastreams are not physical individual files, but actually part of the object file. These datastreams must be obtained from the object file before streaming them back. Inline datastreams are obtained, stored in memory, and streamed back.

Managed datastreams are physical individual files and are handled on physical disk and not in memory. The managed datastream is located in the Fedora file system and streamed from that location. Once the streaming of that file finishes the next datastream in the series is processed.

Performance

Streaming back the package while it is being created has its advantages. The first advantage is that the request is almost immediately acted upon and the package starts being sent. The downside is that data is not continuously being streamed. During the segments of time the next file is being obtained the stream remains open, but no data is being transferred. This is especially noticeable when handling large datastreams. It may take seconds, or even longer, to obtain 100+ megabyte or gigabyte datastreams. During that time the stream download speed will drop-off, since no data is being transferred.



Issues with the ZIP Archiver and PHP's ZipArchive() Class and Zlib Library

ZipArchive() is a packaged class that uses the libzip C library for creating ZIP files.

ZLib is a packaged module that can be used to create GZIP files.

File Size Limitations

Depending on how the packaging service is configured we may want to build packages that are larger than 4GB in size. As of September 3rd, 2012 we are using PHP Version 5.3.6 and the ZipArchive() class with that version of PHP has a file size limit of 4GB. The issue appears to be with the 0.9.1 version of libzip, the C library used for reading, writing and modifying zip archives. With version 5.4.5 of PHP the libzip library is upgraded to version 10.1. The following bug tracks the issue, which is closed as of PHP 5.4.5.

Source: https://bugs.php.net/bug.php?id=51353

The Zlib module can be used to make GZIP files. This module is not hindered by file size limitations; however Zlib only provides in-memory compression and GZIP creation. Files being processed would need to be read into memory,

compressed and encoded as a gzip file, and then streamed when using the Zlib library. Loading extremely large files into memory is not desirable.

Source: http://us2.php.net/manual/en/book.zlib.php

After doing some investigation certain Windows systems have shown that ZIP packages over 2GB will report as being corrupt. This seems to be true for Windows operating systems prior to Windows Vista. Microsoft moved to using the DEFLATE64 algorithm in Vista which supports larger files. Since we are attempting to implement a feature that works on a majority of systems it is not advisable to use ZIP as the archiving schema.

Source: http://support.microsoft.com/kb/301325

Solution

While it would be desirable to use the delivered ZipArchive() or Zlib() solutions PHP provides, the current limitations of the software and some of our users inability to handle large packages forces us to use the TAR archiving standard.

The TAR packaging system does not exhibit the same file size limitations that ZIP does. TAR files are recognized natively by the *nix systems, however they are not by Windows operating systems. Nevertheless, many free to use applications exist that will work in various Windows operating systems.

Datastream Compression

Would it be useful to compress the individual datastreams before returning them as part of a package?

When analyzing the presentation datastreams that could be returned as part of a typical package, it was found compression might not be beneficial. Since presentation datastreams are usually stored in a compressed format, JPEG, PDF, etc. further compression was found to be minimal. Also, the overhead of compressing the datastream in a temporary location will elongate the download process. System resources, especially CPU usage, would be impacted greatly if files were to be compressed. It is the recommendation of this implementation document that we do not compress package files at this time.

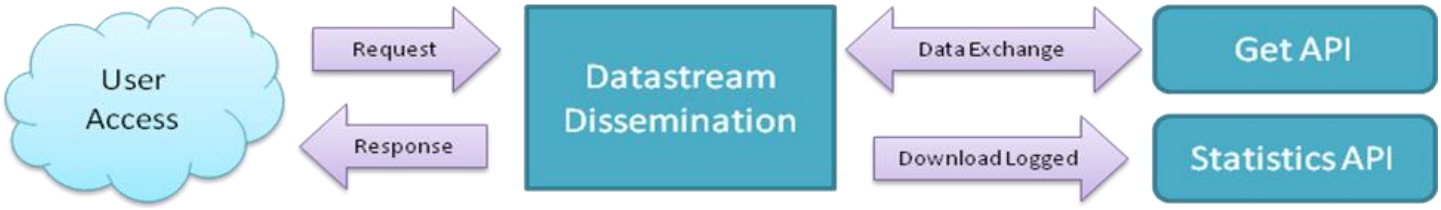
Statistics

When a package is created should datastreams that have been validated and delivered be log individually? If so, where in the process should logging occur?

Currently the responsibility for logging datastream downloads has occurred outside of the Get API. With the inclusion of the new packaging method it might make sense to re-examine the process for logging datastream downloads. Any outside process requesting a package on behalf of a user will not know the true contents of the package being delivered. This could lead to inaccurate logging of datastream downloads.

Making it the responsibility of the Get API's PACKAGE method to report the datastream download would provide us highly accurate download statistics. If we were to take this measure with the PACKAGE method it might make sense to also shift responsibility for individual datastream download logging to the Get API as well.

The following figure represents the current process for logging individual datastream downloads.



The proposal is that for individual datastream downloads the process be transformed, having the Get API log individual download statistics. This proposed process has been outlined in the below figure.

If all responsibility is shifted from the Datastream Dissemination process to the Get API some download statistics will be lost. Currently, we log medium and small JPEG disseminations as downloads, even though no physical datastream exists for those JPEG. Those JPEG's are generated on-the-fly from the JPEG datastream.

For Release 7.0 the Get API will log individual datastream download statistics when a package is requested and delivered. Moving all download statistic responsibility to the Get API will be explored in a future release.

Timeouts

On the development system the PHP max execution time is set to 600 seconds, or ten minutes. Depending on the size of the package and the download speed it is very likely that a package will require more than 10 minutes to deliver.

In the configuration file, lib/configuration.php, two settings have been added.

1) `$cfg_api_get['max_execution_time']` is an integer value, in seconds, that will change the ini max_execution_time setting.

2) `$cfg_api_get['package_reset_max_execution_time']` is a Boolean switch. If set to **TRUE** when individual datastreams are returned the execution time of the script will reset. This allows max_execution_time to be set to a lower value, since it will not have to cover the cumulative package delivery time, but only the delivery times of the individual pieces of the package.