Harvester Service
Technical and User Guide

5 June 2008

THE AUSTRALIAN NATIONAL UNIVERSITY

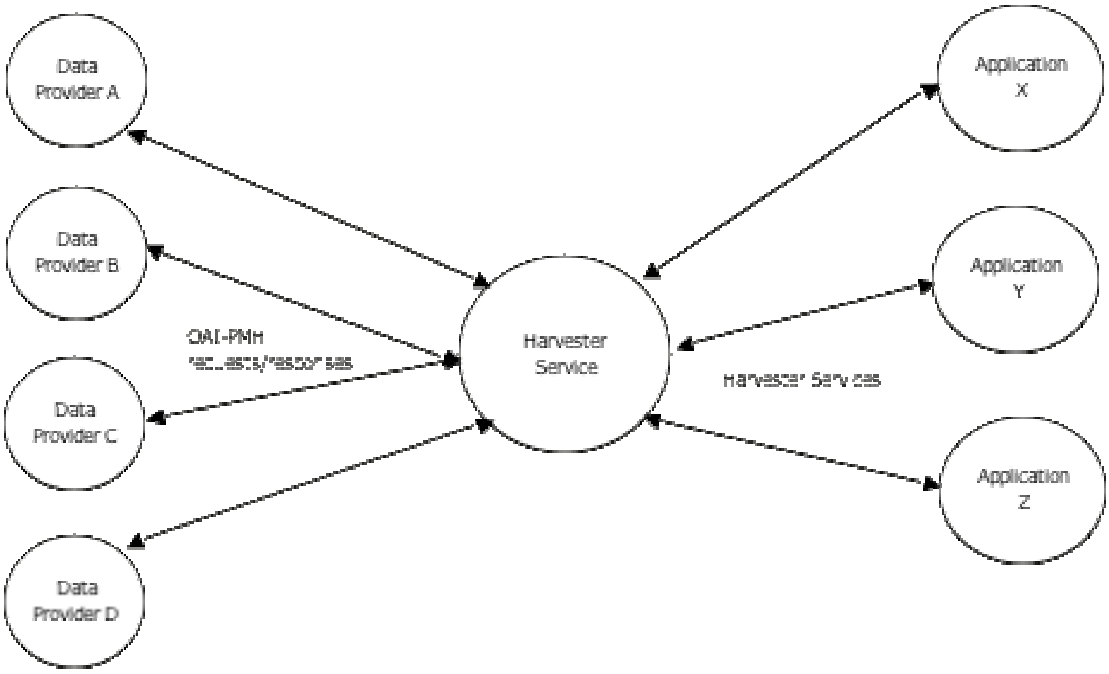
1. Purpose

The purpose of this document is to provide an overview of the Harvester Service package along with technical details of the services implemented.

2. Overview

Note:The Harvester Service is not a software package for end users. It does not come packaged with a friendly user interface. It is a proxy harvester for processing and routing OAI-PMH Data Provider responses to various applications. It is intended it be used for integration with other applications requiring a harvesting service.

The Harvester Service provides a service-oriented framework to support the processing and routing of content and metadata from a source data provider to a target application. It potentially makes management of distributed harvests simpler by providing a single harvest application which can service many clients wishing to perform harvesting without the overhead of writing their own embedded harvester. (Of course the code could also be taken and modified to work as an embedded harvester as well.) A typical deployment configuration for example may be:



In the above diagram multiple applications make use of the Harvester to obtain records from OAI-PMH Data Providers and use the Harvester Service to schedule and run the harvest. For example Application X and Application Y may both harvest Data Provider A each asking for the same or different set of records. In the simplest scenario, upon receiving a harvest request from an external application (which may be a recurring or one-off harvest) the harvester will schedule the harvest for execution. The responses from the data provider are passed back to the application for processing via a service point the application provides in its harvest request. By default a response is forwarded as soon as the harvester receives it. The harvester also is able to cater for custom harvests whereby some or all of the standard harvesting process can be altered to suit particular client applications.

The Harvester Service distribution comprises a number of default harvests including a GET-type harvest (retrieval of XML content from a URL), a standard OAI-PMH harvest, and a custom harvest known as RIF which was developed for harvesting on behalf of a collection/services registry application known as ORCA.

From a software perspective the goals of the Harvester Service development were:

- to make the application lightweight but flexible;
- not to be a burden for IT support staff (and other developers) to maintain;
- to use common and stable technologies;
- to be platform-independent.

The software was written in Java to aid platform-independence and is bundled with a Postgres database.

3. Services

Service: **requestHarvest**

Parameters:

harvestid (Mandatory)

The harvestid is provided by the requesting application and needs to be unique within the harvester. A UUID for example would make a good harvestid. If the harvestid already exists in the harvester an error response will be returned.

sourceurl (Mandatory)

This is typically the base URL of the OAI PMH Data Provider the application wishes to have harvested.

responsetargeturl (Mandatory)

A URL for a service to which the harvester will post its harvested fragments. This would typically be a service on a client application that will process the fragments.

mode (Optional)

Valid values are "test" or "harvest". A mode of “test” returns only the first ListRecords fragment, treated as a one-off harvest and deleted once complete. Default is value is "harvest".

method (Optional)

The harvest method to use. This method must correlate to a class in the thread package with name of form {$method}HarvestThread e.g. PMHHarvestThread. Default is PMH.

metadataPrefix (Optional)

OAI PMH metadataPrefix to use for the harvest. Default is oai_dc.

from (Optional)

Used for date range harvesting. Harvest records "from" this date. Granularities supported are YYYY-MM-DD and YYYY-MM-DDTHH:mm:ssZ. Default is the value taken from the “earliestDatestamp” from the Identify request.

until (Optional)

Used for date range harvesting. Harvest records "until" this date. Granularities supported are YYYY-MM-DD and YYYY-MM-DDTHH:mm:ssZ. Default is the time the harvest commenced.

set (Optional)

A set spec defining a subset of records. Default is null.

date (Optional)

A UTC date in the form YY-MM-DDThh:mm:ssZ indicating the date/time at which the harvest should run. Default is the date the requestHarvest is received by the Harvester.

frequency (Optional)

The recurrence period for a harvest. Currently supported are hourly, daily, weekly, fortnightly, monthly. If frequency is not provided but date is, the harvest is scheduled for one-off execution at the specified date. If frequency is specified but date is not, the time the request was processed will be used as the basis for periodic scheduling. If both date and frequency are not provided the harvest will be executed immediately and treated as a one-off harvest.

Description: The requestHarvest service is responsible for accepting a request from a client to register and schedule a harvest. On receipt of a request the harvest details are set and stored in the Harvester database and the harvest is then scheduled according to the information the

client has provided. Should any errors occur, an XML response will be returned with a reason for the failed request similar to:

```
<?xml version="1.0" encoding="UTF-8"?>
<response type="failure">
   <timestamp>2008-05-07T04:31:32Z</timestamp>
   <message>Missing parameter: responsetargeturl</message>
</response>
```

If the request is successful an XML response will be returned:

```
<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
   <properties>
      <property name="sourceurl" value="http://localhost:8080/dp/request"/>
      <property name="responsetargeturl" value="http://localhost/test.php"/>
      <property name="mode" value="harvest"/>
      <property name="harvestid" value="test"/>
      <property name="method" value="PMH"/>
   </properties>
   <timestamp>2008-05-07T04:32:51Z</timestamp>
   <message>Harvest has been scheduled</message>
</response>
```

The status of the harvest can be checked at any time via the getHarvestStatus service.

Service: **getHarvestStatus**
Parameters:
harvestid (Mandatory)
The id of the harvest
Description: The getHarvestStatus service shows the current status of a scheduled harvest. The response is an XML formatted message:

```
?xml version="1.0" encoding="UTF-8"?>
<response type="success">
   <properties>
      <property name="sourceurl" value="http://localhost:8080/ dp/request"/>
      <property name="responsetargeturl" value="http://localhost/test.php"/>
      <property name="mode" value="harvest"/>
      <property name="harvestid" value="test"/>
      <property name="method" value="PMH"/>
   </properties>
   <timestamp>2008-05-07T04:38:02Z</timestamp>
   <message>Scheduled for 2008-05-07T05:20:14Z</message>
</response>
```

If the harvestid is not found (for example if a one-off harvest has completed and been removed from the Harvester records) an error response similar to that described in the requestHarvest service will be returned.

Service: **deleteHarvestRequest**

Parameters:

harvestid (Mandatory)

The id of the harvest to delete

Description: The deleteHarvest service deletes an existing harvest. The response is an XML formatted message:

```
<?xml version="1.0" encoding="UTF-8"?>
<response type="success">
   <timestamp>2008-05-07T15:53:51Z</timestamp>
   <message>Harvest test deleted</message>
</response>
```

If the harvestid is not found (for example if a one-off harvest has completed and been removed from the Harvester records) an error response similar to that described in the requestHarvest service will be returned. If a harvest is running when the delete is performed the harvest will error. The process for cleanly deleting a harvest is to first use the stopHarvest service before attempting a deletion. Deleting a harvest removes all trace of the harvest from the database.

Service: **startHarvest**

Parameters:

harvestid (Mandatory)

The harvestid of the harvest to apply the service to.

Description: Starts the harvest identified by the harvestid. If the harvest is already running this service will have no effect and the XML response will contain a message to this effect. A harvest must be stopped (either by user or in error) before the startHarvest service will run successfully.

Service: **stopHarvest**

Parameters:

harvestid (Mandatory)

The harvestid of the harvest to apply the service to.

Description: Stops the harvest identified by the harvestid. If the harvest is already stopped or unable to be stopped this service will have no effect and the XML response will contain a message to this effect.

4. Custom Harvests

Custom harvests can be added to the Harvester through the creation of a Java class extending the HarvestThread class. Each HarvestThread class implements a particular harvesting method. For example the GETHarvestThread implements a pseudo-harvest which simply reads a URL pointing to an XML document containing metadata of interest and passes to content direct to an application. The RIFHarvestThread is a custom harvest which performs an OAI PMH harvest but strips the PMH markup and removes namespaces from the metadata payload before forwarding the resultant fragment to the client application.

The only requirements for custom harvests are that the class files must be deployed as part of the au.edu.apsr.harvester.thread package; and to be used (instantiated) the value of the "method" attribute of the requestHarvest service must be the string preceding "HarvestThread" in the class name i.e. {$method}HarvestThread. Custom harvests can override or ignore any harvest details. For example, a custom harvest may ever only need to use one metadataPrefix value and so can be hardcoded within the custom harvest thus overriding the default; it may not want to store response records in the database; it may want to delete a harvest from the system whenever a harvest fails; etc.

5. Notes on Harvest Flow

The harvester is designed to act as a service for applications wanting to harvest from data providers. Typically a client application will send a requestHarvest service request to the Harvester which will result in the Harvester recording the harvest details in its database and then scheduling the harvest for execution. On execution a harvest occurs (the processing and flow is determined by the particular {$method}HarvestThread class) and once completed is either deleted from the Harvester (in the case of a one-off harvest) or rescheduled as an incremental harvest based on the date and frequency provided in the initial client request. Individual response fragments in all current HarvestThread implementations are stored in the Harvester database and also posted back to the URL specified in the reponsetargeturl. When successfully posted are removed from the Harvester database.

In the event the Tomcat servlet container is shutdown all jobs are rescheduled on restart. One-off jobs will be executed immediately on restart.

In the event either a sourceurl or responsetargeturl is unavailable or returns an error, the harvest will retry up to three times and then fail with an error status and rescheduled based on the harvest record's date and frequency. The harvest will need to be restarted manually via the manage service if needing to rerun prior to the next scheduled run.

6. Source Code Overview

A brief overview of the source organisation follows. For further details individual code modules should be examined. A javadoc option with the build file is available (use ant target of 'javadoc').

harvester->src

Java Source Code

harvester->docs

Install guide, license and the like can be found in the docs area.

harvester->etc

web.xml file and Postgres database DDL

harvester->lib

All external jars and associated license details.

harvester->stylesheet

XSLT Source Code

harvester->src->...->dao

DAO classes for Postgres database actions. These classes can be substituted for classes supporting other database platforms.

harvester->src->...->oai

OCLC's Harvester2 verb classes.

harvester->src->...->servlet

Classes implementing the services described earlier in this document

harvester->src->...->testing

Classes used in development for testing which may be of use to other developers.

harvester->src->...->thread
Classes implementing harvests and the harvest manager class.

harvester->src->...->to
Transfer object classes. These classes define an interface between client and business classes and the DAO classes.

harvester->src->...->util
Classes containing methods useful to numerous other classes and XSL Java extension methods.