AddOnMechanism 2fAddOnPrototype

<?xml version="1.0" encoding="utf-8"?>
<html>
AddOnMechanism > Add``On Mechanism Prototype

AddOn Mechanism Prototype

There is an initial prototype of an Add``On Mechanism proposed for DSpace currently available in CVS. This can be obtained from the Tapir Source``Forge page: http://sourceforge.net/projects/tapir-eul
There are two modules that are important:

- Add``On - This module contains a skeleton add-on that supported DSpace add-ons should be implemented with. It consists of a standardised build file which needs to be filled out to meet the needs of the add-on, and a directory structure to which add-ons should conform. Each directory contains a EADME file with scope notes to aid population. In addition, there is a dspace directory which contains proposed additions and changes to the current DSpace core to support the add-on process. Currently this consists only of a new build file which is capable of delegating to the add-on's own build file in a structured way, and a class to import required DC metadata fields to the DC type registry. Further additions planned include changes to the way that CSS files are managed and the addition of a org.dspace.addon package which will contain the tools that the new build file will implement to aid installation of add-ons, such as building config files and so forth.
- Tapir-1 - This module contains the new version of Tapir which will eventually be released as v1.0. This add-on is being developed alongside the Add``On Mechanism as an example of implementation and a driver for functionality. At time of writing the core java, the JSPs, the database tables and the required DC fields can be installed using the build mechanism in the Add``On module. Current challenges are to smooth over the process of configuration, which will no doubt form the hardest part of the Add``On Mechanism. No claim is made at this stage that the Tapir will actually function correctly once installed.
  Access to these modules can be obtained via anonymous CVS access using the CVSOOT: {`/cvsroot/tapir-eul`}. Write access to the repository will be made available to current DSpace committers if desired. Patches and suggestions are welcome and encouraged. If you have an add-on that you would like to produce as an experiment with this mechanism please do so, and use the dspace-devel mailing list to collaborate with others.
  Note: This Add``On Mechanism supports only the installation of one add-on at a time using a syntax like:
  {{
  Unknown macro: {% ant -Dinstall=/path/to/addon addon}}
  }
  Updates on the work available at the bottom of this page...

Java Source

This Add``On Mechanism does not support patching the DSpace source code. The intention is that all new add-ons will provide (if necessary) their own java source and build it into their own jar file. The skeleton add-on build file contains examples of how to do this. This jar file will then be included by the DSpace build file into the standard library.
The reason for this decision is that patching the source is not technically adding *on*, but rather adding *in*, which will produce maintenance problems in the long term which may be difficult to resolve. We are therefore encouraging add-on writers to make their code stand as separately as possible from the DSpace core (although obviously it can be built against the dspace library).

JSP Files

Currently the Add``On Mechanism does not attempt to merge add-on JSPs with existing DSpace JSPs, and may well never do so. Instead all JSPs are simply copied into the {`dspace/jsp/local`} directory for deployment along with all other localisations. This means that there may be clashes with existing localisations, or other add-ons, and there is currently no mechanism to deal with this. Suggestions welcome for the best way to deal with this. The initial idea was to produce an Add``On Mechanism to aid the installation of third party tools, but not necessarily to guarantee success. Perhaps at this stage a simple warning process would be warranted.

Database tables

Aside from ob's suggestions regarding good practice in making schema changes (i.e. do not modify core DSpace tables, and ensure that add-on tables are namespaced appropriately), this mechanism delegates all tasks of creating database tables to the add-on's build file. The recommended method of making schema changes is to use the Initialize``Database class in the DSpace core, and an example of usage is in the skeleton add-on build file. This Add``On mechanism does not verify that the commands being issued are in keeping with being a well-behaved add-on.

Configuration Files

Dealing with configuration is one of the main challenges for the Add``On Mechanism. The proposed solution is to produce a pre-installation config directory: {`dspace-source/build/config`}. The instance of DSpace having an add-on installed will need to be installed alone initially, and this will produce the file {`dspace-source/build/config/dspace.cfg`} which will be a straight copy of {`dspace-source/config`}. When the add-on is installed through the DSpace build file, the add-on's config file in {`addon-source/config/dspace.cfg`} will be combined with {`dspace-source/build/config/dspace.cfg`}, producing a new configuration without damaging the clean DSpace config file. This should make uninstallation or upgrades of add-ons possible without too much difficulty. The exact build process needs to be carefully defined (TODO) in order to ensure that the correct config files are always in use. This will result in a change in the way that config files are managed in DSpace, and may place a higher dependency on always having the source code available.
This process ought to be possible also with configuration such as log4j.properties, dspace-web.xml, oai-web.xml, and dspace-tags.tld, although it will be slightly more complicated than the dspace.cfg solution.
We also need to be careful to define exactly which files the Add``On Mechanism can deal with. For example, input-forms.xml may be a configuration file, but may ultimately be part of a submission module which cannot be reliably dealt with by an Add``On Mechanism, in the case

that perhaps the module is unavailable or is not in use in preference for another. This introduces a consideration of a dependency chain for add-ons which is beyond the scope of getting a basic Add``On mechanism functioning at this stage.

AddOn egistry

(See Add``On egistration below for more concrete thoughts)
We suggest that each add-on should be registered somewhere in the live DSpace instance so that subsequent upgrades or uninstalls can be managed. The exact form of this registry is currently unclear, but perhaps a directory {`dspace/addons`} would be appropriate, containing XML files describing all of the currently available add-ons, with version information and perhaps file lists and so forth (a thought occurs that compatible versions of DSpace might be listed too).
An alternative solution is to include a list of installed add-ons into the dspace.cfg file either during manual configuration or automatically at build time.
Given potential concerns with the sensitivity to the order in which add-ons are applied, it may also be sensible to include some information regarding installation order.

XML UI

Just to add a hook regarding ob's point about the XML UI; exactly how this code will impact the Add``On Mechanism is unclear (to me, at least). I suspect that some kind of method of inserting new data into the Site``Map will be required, and that much of the underlying moving about and compiling of code will remain the same.

Some Further Suggestions

The add-on manager sits outside a DSpace source tree and assembles a new source tree that can be built in the usual way:
{{

Unknown macro: {myDSpace/myDSpace/dspace1.3.2myDSpace/tapirmyDSpace/checker}}

}
unning ant in {`myDSpace/`} creates an integrated-dspace directory with the patched sources. Going into integrated-dspace gives you the current build environment. You can update each plugin / main source independently and the Add``On manager will still work (but won't give you any assurances about the new permutation working)

AddOn Update and Removal

Update and removal processes are unclear at this stage. We have included an example update script in the skeleton add-on, although exactly how it is implemented is not certain. One suggestion is that there is a single update script after the initial create script which contains all schema changes as ALTE commands. This means that you should be able to update to the latest version from any previous version with one single file. There may still be a necessity, though, to run update scripts to modify database contents, for example, which could cause problems. If each update is really an installation from scratch but with additional schema changes then that would solve any problems with regard to functionality being dropped (e.g. a servlet mapping is no longer required in the web.xml).
If we have a good registry of add-ons, it should be possible either to rebuild the entire system from scratch without the module selected for removal, or to surgically remove all references to the old module. The latter may not work. We should require that new add-ons come with database destroy as well as create scripts.

AddOn Registration

In order to keep track of add-ons, it would probably be best to register them in the live dspace directory: {`dspace/addon/addon-name/`}, where {`addon-name`} comes from the {`addon.properties`} file in the source. I think that the registry needs to contain the following information:

- List of JSPs
- List of external jar files
- Database destroy script
- a reference to the source code directory, or perhaps better, the source tree itself
- XPaths for installed XML bits (not quite sure how this will work yet)
- An uninstall java class, perhaps?
- The current version number

With this information it should be possible to automatically uninstall the addon.
Some thoughts with regard to how this directory might be structured then:
{{

Unknown macro: {addon-name/info.xml # the addon description, as described below/src # the full source directory of the addon/uninstall # the directory containing uninstall scripts}}

}
The {`info.xml`} file would then be a full description of the addon, so that its parts can be identified for removal or upgrade or whatever. It may look something like this:
{{

Unknown macro: {<addon name="addon-name" version="1.0"><uninstall>org.dspace.administer.addon.Uninstall</uninstall><jsps><jsp path="relative/path/to/jsp" /></jsps><jars><jar name="external.jar" /></jars></addon>}}

}
The idea is that the DSpace build time Add``On Mech will construct this file itself, rather than asking the add-on authors to do it. The {`<uninstall>`} tag suggests a class that the add-on authors would provide which performs any maintenance operations on uninstall. {`org.dspace.administer.addon.Uninstall`} is a suggested interface for the uninstall class which would be implemented by the add-on.
On removal, the jsps and jars that this add-on relies on could be cross checked with other add-ons to ensure that no damage will be caused at uninstallation. For this reason it may also be useful to have a dspace info.xml file, which would prevent the removal of any core components (this will probably only really apply to jar files).

Update 11/11/2005

I have had a go at the build time configuration process, and have come up with the following procedure:
If {`fresh_install`} is requested:
1) Move all source configs from {`dspace-source/config`} to {`dspace-source/build/config`}
2) Copy the file referenced by {{{ $

Unknown macro: {config}

}}} into {{{ dspace-source/build/config }}} in case it is not the default
3) {`init_configs`} target moves the contents of {`dspace-source/build/config`} to {`dspace/config`}
4) {`ConfigurationManager -installTemplate`} still operates on {`dspace/config`}
If {`addon`} is requested:
1) Add``On``Configurator is called which opens {`addon-source/config/dspace.cfg`} and appends it to {`dspace-source/build/config/dspace.cfg`} along with a note to edit the source config, not the live config
2) The target update is called. This will need to move all new config files into the live directory, which may require {`ConfigurationManager -installTemplate`} to be run again
Other thoughts:

- some add-ons will come with additional configuration files. These will need to be moved into {`dspace-source/build/config`} and then copied over during update
- the whole process will eventually rely on all the sources being available. Perhaps the very first thing to do would be to copy the add-on source into the dspace source tree somewhere, and build from that version. At least then folk will only have to be sure to keep the one top level directory intact. Or perhaps they could all be stored in {`dspace/addon`} as full sources (perhaps the dspace source tree needs to be maintained this way too?)
  Unfortunately the code to do the above processes is stuck on my computer until I can get in contact with the CVS server again. Sometimes getting onto SF during the afternoon in Northern Europe is pretty difficult. I will upload it at the start of next week. In the mean time attached is a tarball with the current state of the Add``On package at my end: attachment:addon-2005-11-11.tar.gz

</html>