Creating an SLC Data Provider

This wiki page describes how to create a new VMS hosted AIDA Data Provider.

See also the main page describing development of VMS hosted AIDA Data Providers, that page covers the overall architecture and revolving maintenance of data providers. This one covers only the process of creating the files which implement a brand new data provider.

To create a new data provider on SLC, we begin on an AFS unix node and use facilities in the Aida makefile system for creating the CORBA stubs and skeletons of a new Aida server. We then copy all the files so created over to VMS and continue real development there (where real developers work):

First, create a development directory on AFS unix, and cvs checkout aida:

```
$ mkdir work2
$ cd work2
$ cvs co package/aida
cvs checkout: Updating package/aida
U package/aida/.classpath....
```

Source the Aida development setup files. Note the 2nd argument to aidaSetEnvDev.csh is the directory of your development instance of aida (what you created in the previous step):

```
$ source /afs/slac/g/cd/soft/dev/script/ENVS.csh
$ source /afs/slac/g/cd/soft/dev/script/aidaSetEnvDev.csh dev ${HOME}/work2/package/aida
```

cd to the dp directory

```
$ cd package/aida/edu/stanford/slac/aida/dp/
```

Create a new Data Provider (dp) directory. Then copy an existing Makefile.sun4 from one of the existing Data Provider directories, on which we'll base this new one.

```
$ mkdir dpSlcBpm
$ cd dpSlcBpm/
$ cp ../dpSlcModel/Makefile.sun4 .
```

Modify the Makefile.sun4 file, replacing the PACKAGE, MODULE and INTERFACENAME values with values appropriate for your new data provider.

```
$ emacs Makefile.sun4
<edit PACKAGE, MODULE and INTERFACENAME>
```

Run the Makefile.sun4; this will create all client and server sides, except the _impl.java file. You will create an _impl.java file by hand later.Ignore the jidl complaint about not finding an idl file, it's not supposed to.

```
$ gmake -f Makefile.sun4
jidl: couldn't open /u/cd/xxx/work2/package/aida/idl/dpSlcBpm.idl
sed -e 's/THIS/dpSlcBpm/g' ....
...
```

Go to VMS, and "create the usual 43 layers of Aida directory structure", to the .DP directory. Then create your new subdirectory underneath it.

```
> set def [.work.package.aida.edu.stanford.slac.aida.dp]
> create/dir [.dpslcbpm]
> set def [.dpslcbpm]
```

FTP the java files you created on AFS, to your VMS development directory. You can execute the ftp command from unix or from VMS (in which case ftp to ftp.slac.stanford.edu). Here we have doen the ftp from unix. Eg:

```
> sftp mccdev
Connecting to mccdev...

sftp> cd /data_disk_slc/slc/greg
sftp> cd dev/aida/package/aida/edu/stanford/slac/aida/dp/dpslcbuffacq
sftp> mput *.java
```

Back on VMS, purloin a *server.java file from one of the existing VMS data providers (eg ref_aidashr:dpslcmodelserver.java). Copy it to your working directory and modify it. Also copy an existing *_impl.java file as a starting off point for your new server.

```
> copy ref_aidashr:dpslcmodelserver.java dpslcbpmserver.java
> copy ref_aidashr:dpslcmodeli_impl.java dpslcbpm.java <make your changes to the impl file in particular>
```

Define the development classpath, so it uses your development directory first:

```
define myjava$classpath [------]
```

Compile your java code:

```
javac -g *.java
```

Create the JNI prototypes for the native methods:

```
MCCDEV> javah -o dpslcbpm_jni.h -classpath [------] "edu.stanford.slac.aida.dp.d
pSlcBpm.DpSlcBpmI_impl"
```

Create your JNI interface module (the C code that the java calls). By convention we call this module dp<servername>_jni.c. It must have the ***ATTRIBUTES** *=JNI at the top so cmp knows how to compile it, and it should include the output file from the step above. Eg:

```
/*
      **MEMBER**=SLCLIBS:AIDASHRLIB
      **ATTRIBUTES**=JNI
*/
#include "dpslcbpm_jni.h"
```

Create the dp<servername>_jni_helper.c file. This is the module that defines the functions that access the control system. The names of the functions it *defines* must all be in upper-case (due to the compiling and linking configurations specified in slccom:compile.com and the build com files). But the functions that those function call (in the control system shareables), need not be in upper-case.

Compile the dp<servername>_jni.c and dp<servername>_jni_helper.c file into an aidashr_devlib.olb

```
MCCDEV> lib/create aidashr_devlib.olb
MCCDEV> cinc *.c
```

Create an aidashr_xfr_alpha.opt. Due to namemanging of JNI and VMS linking
namespace restrictions, it's important to do this with java$jni_example:scan_globals_for_option.com.

```
MCCDEV> libr/extract=dpslcbpm_jni/output=dpslcbpm_jni aidashr_devlib
MCCDEV>  @java$jni_example:scan_globals_for_option *.obj aidashr_xfr_alpha.opt
```

Link your aidashr, using the opt file created above. You may also need to use a locally modified version of AIDASHR_XFR_ALPHA.OPT if you are using shareables AIDA hasn't used before.

```
MCCDEV> buildtest aidashr /default
```

Add the server to the list of Aida servers. This list is part of the Aida directory service Oracle database. See the Aida Directory Database Guide for details, off the Aida homepage. See the Aida SQL Cheatsheet, Adding a new Data Provider.

```
SQL> @/afs/slac/g/cd/soft/ref/package/aida/common/script/add_service 'SLCBpm' 'SLC BPM orbit acquisition'
```

Add instances and their attributes for testing. Here's a single example that may be one of thousands when it comes to deploying your server:

```
SQL>  @/afs/slac/g/cd/soft/ref/package/aida/common/script/add_IA 103 'P2BPMLER' 'BPMS'
```

Clone a server config file, that defines on which port this server will run. The config filename must be the same as the servername (see the java command line in the START<dpname>.COM file (see below)).

```
copy ref_aidashr:dpslcmodel.conf dpslcbpm.conf
<edit the file and choose a unique port number>
```

Run the server. The java command to run an AIDA server takes several arguments, to pass the values of environment variables to the server at startup, so we have little .com files for each server. To override the default classpath (JAVA$CLASSPATH) which points to the aida.jar in slcimage, and the default AIDASHR (in SLCSHR), redefine the logicals before running the START<dpname>.COM file:

```
MCCDEV> define myjava$classpath [------]
MCCDEV> testshrx/define/default aidashr
MCCDEV> @startdpslcbpm
```

Release

Release is described in "release" section of the SLC Peer Programmers Guide

SLC Peer Programmers Guide

When development is complete on VMS side

When you're done, you can cvs release the unix side aida checkout. Note that you do not need to checkin any of the CORBA stubs and skeletons you created for the VMS server. The client side of aida sees the VMS server simply as an "aidaObject", it does not know it is specifically talking to a dpSlcMagnet server object, so there is no need to keep any of the files for a VMS server on the unix side.