ProServer Guide

Contents

Server Features

One of the strengths of the Distributed Annotation System (DAS) is its 'dumb server, clever client' architecture, which in theory allows even research groups with limited informatics resources to provide distributed access to their data. ProServer attempts to realise this strength by providing a framework for hosting data via DAS that is:

- lightweight
- stable
- easy to install
- easy to extend
- free
- open source

However, ProServer also offers some features beyond the core DAS specification that allow it to be used in a complex high performance environment.

Multitasking

ProServer is a standalone forking HTTP server based upon the Perl Object Environment (POE), a framework for event-driven multitasking applications in Perl. Using this framework, the server distributes concurrent requests between several child instances. The maximum number of processes ProServer uses (and therefore the maximum number of simultaneous client requests) is configurable, allowing a balance to be struck between resource usage and performance.

Compression

Where supported by clients, ProServer will reduce the size of lengthy responses using GNU Zip (Gzip). Clients wishing to take advantage of this support should set the standard 'Accept-Encoding: gzip' HTTP header (most web browsers do this).

Flexible Deployment

Being a standalone program, ProServer can be quickly deployed on any machine, and does not rely on an HTTP server such as Apache. It is also flexible enough to be integrated into existing webserver architectures. Both the Sanger Institute and European Bioinformatics Institute use ProServer in a load balancing, reverse proxied, clustered configuration.

XSL Stylesheets

The Extensible Stylesheet Language (XSL) defines how XML documents may be transformed into other formats. ProServer provides XSL Transformation (XSLT) stylesheets which clients such as web browsers use to present XML data in formats more amenable to human consumption (rather than for computer consumption, for which XML was principally designed).

You may consider modifying or adding to these default stylesheets if you wish to:

- apply a site-wide style to ProServer's pages
- change how features from a particular source are presented

Currently, ProServer supplies XSLT stylesheets for the features, sources and dsn commands. Stylesheets for other commands are under development.

DAS Extensions

ProServer v2.7 is a server implementation of the DAS/1.53E specification. This is an extended version of the current 1.53 DAS version. The 1.53 specification is published on the BioDAS website, and the 1.53E extensions are published at the DAS Registry.

In brief, the 1.53E extensions comprise:

- The 'sources' server command
- The 'alignment' source command
- The 'structure' source command
- The 'interaction' source command
- The 'volmap' source command
- The 'GeneDAS' extension
- The 'maxbins' features command parameter
- Stylesheet conventions
- Timestamp conventions
- An ontology

Installing ProServer

ProServer is hosted at the Wellcome Trust Sanger Institute, and is also available for download from CPAN.

Downloading

Check out the code from the Sanger CVS server using the "cvsuser" account. The password is "CVSUSER" (without the quotes):

```
[bash] cvs -d :pserver:cvsuser@cvs.sanger.ac.uk:/cvsroot/Bio-Das-ProServer login
[bash] cvs -d :pserver:cvsuser@cvs.sanger.ac.uk:/cvsroot/Bio-Das-ProServer checkout Bio-Das-ProServer
```

ProServer is also distributed as a package via CPAN. It is available for download at the website or using the CPAN command-line utility.

Building

ProServer is designed to be automatically built using the *make* utility:

```
cd Bio-Das-ProServer
perl Makefile.PL
make
make test
```

You may optionally install the ProServer modules into your Perl distribution:

```
make install
```

ProServer has a small number of standard dependencies (many of which you should already have). Any missing dependencies will be reported, in which case you should install them from CPAN. See the included README file for a list of required modules.

If any of your DAS sources will connect to a database, you will also need the *DBI* module and relevant driver (e.g. *DBD::mysql* or *DBD::Oracle*).

Running

ProServer is distributed with an example command-line executable in the *eg* directory:

```
eg/proserver --help
```

The default proserver.ini file contains details of the configuration options the server understands. Also see the INI Format section for details of configuring DAS sources.

An example CGI script is also provided in the *eg* directory.

Designing a DAS Source

The first step in exposing your data using DAS is determining how your data is to be offered. Consider some of these points when designing your DAS source to maximise its accessibility and usefulness.

Co-ordinate Systems

Is your data based on genomic co-ordinates, Ensembl Gene ID's, or perhaps proprietary identifiers? Are your reference sequences from the latest assembly?

If possible, it is best to expose your data on the most recent version of an assembly/database. However, if you have conducted some form of sequence analysis on an old or modified version of a sequence, you may need to define your own co-ordinate system.

Services

You probably want to expose features of some form, but could you also define a stylesheet to govern the display in clients such as Ensembl or SPICE? If your reference co-ordinate system is unique or not widely used you should provide a reference source offering sequences and entry points, but could you offer mapping alignments with segments from another co-ordinate system?

It is also very useful for your DAS source to be capable of informing clients of the valid entry points you source can annotate, or the types of features. This is especially true for DAS sources with numerous features or many types of features.

Intended Usage

Is your data purely display-driven, and if so which clients will it be compatible with? Ensembl, SPICE, Dasty and Pfam are all graphical DAS clients you may consider testing your source with. Is your data amenable to being used programatically? Consider fleshing out your features with terms from the 1.53E ontology created for the BioSapiens project.

Data Storage

For small numbers of features, a flat file may be a sufficiently fast storage medium for your data, but often an indexed relational database is necessary. If you have an existing database, does it contain details of your reference sequences such as versions or checksums? ProServer can take a lot of the work out of making your data as useful as possible if you store some of this information:

- A list of the reference segments used to derive your features.
- The length of each reference segment.
- The version (often a checksum) of each segment.

Implementing a DAS Source

This section is a developers' guide to implementing a DAS source, intended as a companion to ProServer's POD documentation. It is assumed that you are somewhat familiar with the concept and basic architecture of DAS, which you can read about on the BioDAS website. ProServer also supports the 1.53E extensions as described at the DAS Registry. Writing a custom DAS source requires intermediate object-oriented Perl programming ability.

ProServer is designed to be a lightweight DAS server that is simple to set up extend. Each server can host one or more DAS sources, with each source (or DSN) being represented by a single *SourceAdaptor* Perl object and INI configuration. Implementing a DAS source in ProServer therefore entails providing a subclass of the *Bio::Das::ProServer::SourceAdaptor* package and INI to configure it.

INI Format

ProServer takes its configuration from a standard INI file, specified at startup. The file is divided into sections: one 'general' section for server-specific options, and one section per DAS source. The various server-specific options are described in the example *proserver.ini* file. The server processes each other section as follows:

Property	*Example*	*Function*
section	[simple_human]	Required; defines the DAS source name (DSN)
adaptor	adaptor = simpledb	Required; the SourceAdaptor subclass that will represent the source.
state	state = on	Unless set to 'on', the source is not enabled.
transport	transport = dbi	The Transport subclass that will be built for the source.
autodisconnect	autodisconnect = 1800	Specifies that the Transport should clean up after itself following a command. Can be 'yes', or a specified number of seconds.
hydra	hydra = dbi	Specifies a 'multi-headed' Hydra source. A single definition can generate multiple sources.
parent	parent = simple_mouse	Specifies that a source should inherit properties from another source. Only undefined propertiesare inherited. Chained and reciprocal inheritance is permitted.

You may also specify additional custom properties: these are passed into the SourceAdaptor object stack.

Transports

Each SourceAdaptor may be configured with zero or more *transports*. A transport is designed to handle data access implementation, reducing the need to write boilerplate code. ProServer is supplied with several *Bio::Das::ProServer::SourceAdaptor::Transport* implementations, allowing easy access to data sources including, for example, relational databases, flat files and the Ensembl API.

Transports are passed the same INI properties as SourceAdaptors, allowing them to be configured in the same way. For example, the *DBI* transport requires the 'dbname' parameter. See individual transports' POD documentation for details. Below is an example that uses the DBI transport to handle the tedious aspects of querying a relational database.

```
# Generic features stored in an SQL table
my $features = $self->transport->query('select * from features where segment = ? and end >= ? and start <= ?',
                                       $segment, $start, $end);
```

Although most sources have only a single transport, it is possible to configure multiple transports for a single source. This can be done by specifying overriding properties for named transports. This is best illustrated with an example:

```
[foobar]
state         = on
adaptor       = doubledb
transport     = dbi
dbuser        = anonymous
dbname        = foodb
bar.dbname    = bardb

my $foos = $self->transport()->query($sql, @args);      # connects to 'foodb'
my $bars = $self->transport('bar')->query($sql, @args); # connects to 'bardb'
```

Hydras

A hydra source is a 'multi-headed' source with a single configuration. A *Bio::Das::ProServer::SourceHydra* can be used to automatically create several sources, each using the same *SourceAdaptor* implementation. For example, the 'dbi' SourceHydra generates a SourceAdaptor object for each database table matching a given prefix.

Command methods

The Bio::Das::ProServer::SourceAdaptor base package contains much of the code to handle DAS requests and format an appropriate response, with several 'stub methods' left for you to implement. In particular, each DAS command is associated with a 'build' method that SourceAdaptor subclasses should override if it is to implement the command. Each of these methods is called with the arguments given to the command, and expects a specific data structure. Details for arguments and return types are given in the POD documentation for

Bio::Das::ProServer::SourceAdaptor. Some commands also execute other methods which may be optionally overridden.

Implemented commands must also be specified in the 'capabilities' metadata in order to be activated.

Features

Method	build_features
Also calls	init_segments, known_segments, length, segment_version

Types

Method	build_types
Also calls	known_segments, length, segment_version

Sequence; DNA

Method	sequence
Also calls	known_segments, length, segment_version
Notes	The 'segment_version' method is only called if no version is provided in the returned data structure.

Entry Points

Method	build_entry_points
Also calls	-
Notes	Has a default implementation that relies on the 'known_segments' and 'length' methods.

Alignment

Method	build_alignment
Also calls	known_segments

Structure

Method	build_structure
Also calls	known_segments

Volmap

Method	build_volmap
Also calls	known_segments

Interaction

Method	build_interaction
Also calls	-
Notes	Does not filter unknown segments (this command treats query segments differently).

Other methods

These methods are not tied to a single DAS command, but rather may be called in support of several. None are explicitly required for a functioning source, but all make the source more useful (e.g. by providing details of the sequence upon which annotations are based). Therefore it is best to implement as many as possible.

Method	*Purpose*	*Default*	*Note*
known_segments	Implement this method to provide a list of identifiers known to the DAS source, used by ProServer to filter requests for unknown or incorrect segments.	-	By default 'build_entry_points' calls this method.

length	Implement this method to provide the length of a segment as it is known to the source. This is used by ProServer to filter requests for invalid ranges.	0	By default 'build_entry_points' calls this method.
segment_version	Implement this to provide a version or checksum of a segment as known by the source.	1.0	-
init_segments	Purely a convenience, called before build_features to allow the source to prepare the data for a list of segments if this is more efficient.	-	-

Stylesheets

The stylesheet command does not need to be configured in code. Instead, it is resolved using:

1. A 'stylesheet' INI property. The value should be the whole stylesheet XML (inline).
2. A 'stylesheetfile' INI property. The value should be the location of the XML file.
3. The default stylesheet, which draws features as a black box. May be changed by overriding the 'das_stylesheet' method.

XSL Stylesheets

The same technique for defining the stylesheet command also applies to XSL stylesheets. XSL stylesheets are used by web browsers to transform the XML responses of DAS commands into a human-readable format.

Here, the relevant INI properties are 'features_xsl' or 'features_xslfile' etc. Not specifying either results in the default ProServer XSL being used.

Homepages

ProServer provides a default 'homepage' for each DAS source, which gives some simple information about the source. However, it is possible to provide an HTML page to display instead, in the same manner as for stylesheets.

Metadata

Each DAS source should provide information about itself that helps clients to determine what kind of data it offers. In true TMTOWTDI Perl spirit, ProServer provides several ways to provide the metadata, either in code or via INI properties. Note that the 'capabilities' property is *required*.

1. Overriding the relevant method. See the *Bio::Das::ProServer::SourceAdaptor* POD documentation for details.
2. Specifying a config property.
3. Setting a variable in the object stack (using the 'init' method).
4. Nothing: the default value (if any) is used.

Below is a list of metadata properties you should provide for your source:

Property	*Type*	*Purpose*	*Order of preference*	*Default*
capabilities	hashref	Commands and options offered (sources command)	method; stack	-
coordinates	hashref	Co-ordinate systems and test ranges (sources command)	method; stack; ini	-
properties	hashref	Custom tags (sources command)	method; stack	-
title	text	Human readable name (sources command)	method; ini	The source name (DSN)
description	text	Human readable description (sources/dsn command)	method; ini	The title
doc_href	URL	Location of documentation/homepage (sources command)	method; ini	A default ProServer homepage.
source_uri	text	Used to group sources (sources command)	method; ini	The source name (DSN)
version_uri	text	Used to group sources (sources command)	method; ini	The source URI
maintainer	email	Identifies a point of contact (sources command)	method; ini	The server maintainer
dsncreated	date	Source date (sources command, HTTP headers)	method; stack; ini	The 'last modified' date of the Transport (if supported) or epoch

dsnversion	number	Source version (dsn command)	method; stack; ini	1.0
mapmaster	URL	Reference source (dsn command)	method; ini	-

Co-ordinates can be specified in the INI file using the format:

```
coordinates = NCBI_36,Chromosome,Homo sapiens => X:10000000,10111111 ; Ensembl,Gene_ID,Homo sapiens => ENSG00000000001
```

Or in code using:

```
sub init {
  my $self = shift;
  $self->{'coordinates'} = {
    'NCBI_36,Chromosome,Homo sapiens'                      => 'X:10000000,10111111',
    'ensembl,gene_ID,homo sapiens'                         => 'ENSG00000000001',
    'http://www.dasregistry.org/dasregistry/coordsys/CS_DS6' => 'BRAF_HUMAN'
  };
}
```

Here, the key is either the URI or description of the co-ordinate system (see the included registry coordinates XML file for details). It is case insensitive. The value is a segment range that can be used to test the source. See the DAS Registry documentation for more details of co-ordinate systems.

Registration

Many clients, such as Ensembl and SPICE, automatically connect to the DAS Registry to retrieve a list of DAS sources. If you register your source, it will reach a wider audience. The registry can also monitor your DAS source and inform you if it is not working correctly, and also provide an 'auto-activation' URL that will enable and configure your DAS source in Ensembl.

Because registered DAS sources are automatically available to several clients, it is preferable for registered DAS sources to be as 'well-formed' as possible. This includes providing accurate and up-to-date metadata for your source, as well as consistent and usable data. You may wish to consider whether your data fits into the 1.53E ontology developed for BioSapiens.

Examples

There are several SourceAdaptor implementations provided with ProServer that serve as useful examples. The 'simple' adaptors may be particularly useful as starting points.

Updating to DAS/1.53E

If you have already developed DAS sources, you may wish to update them to support the 'sources' command, which provides for more meaningful descriptions of the services that a DAS source offers. Updating your source is a simple matter of providing some metadata: see the Metadata section of the guide for details of how this is done. You will probably want to add the following:

- title
- description
- capabilities
- coordinates
- maintainer
- dsncreated (mysql and file transports provide a default implementation)

Further Information

The following links provide useful background or further information about using DAS:

- http://www.biodas.org
- http://www.biodas.org/documents/spec.html
- http://www.dasregistry.org
- http://www.ensembl.org/info/data/external_data/das/index.html

Bug reports *et cetera* should be directed to roger.pettett@sanger.ac.uk or andy.jenkinson@ebi.ac.uk.