NativeSPSunConfig

Half of Shibboleth runs within the web server. For Sun/iPlanet, this half is implemented in an NSAPI module called **nsapi_shib.so** or **nsapi_shib.dll**, depending on the platform. The initial configuration is controlled with the **magnus.conf** and **obj.conf** configuration file(s), but one of the primary options there (normally unneeded) is to point the module at the overall SP configuration file (**shibboleth2.xml**) where a lot of the options not specific to the web server are controlled.

At runtime, the module has the ability to process both a variety of NSAPI function parameters and rules specified in the SP configuration and make sense of both. This allows for a choice of approaches based on the need for native integration or for portability between web servers.

Loading the Module

Most of the primary options needed to make the module usable are compiled into the code when you build it. This allows the module to initially load in most cases by modifying the **magnus.conf** configuration file as follows (the exact shlib path will vary):

```
Init fn="load-modules" funcs="nsapi_shib_init,nsapi_shib,shib_handler" shlib="C:/opt/shibboleth-sp/lib
/shibboleth/nsapi_shib.dll"
Init fn="nsapi_shib_init" server-name="newman.uts.ohio-state.edu"
```


OS Compatibility

Because both this web server and the SP software include C++ code, the SP module will only load and run if it's built with the same compiler and C/C++ runtime library as the web server itself. Unfortunately, this is only known to be true on Solaris (where Sun's compiler is used) and Windows (where the conflicts aren't a problem because shared libraries work much better there than on Unix).

Typically you'll need to provide the server instance's virtual hostname, as shown, or the module will be forced to try and guess the name. We don't have experience using this server in actual vhost scenarios, but there is untested code that may or may not be able to derive the correct hostname when name-based virtual hosting is done out of a single server instance. This will require thorough testing on your part.

Initialization Options

There are a handful of parameters to the NSAPI Init function that apply to the module's overall configuration and are usually left out in favor of the values generated at compile time. They also correspond to a number of environment variables that can be used in place of commands. They are generally needed only when the software is run out of a different directory from the build path.

`shib-prefix`	Corresponds to SHIBSP_PREFIX variable.
`shib-config`	Corresponds to SHIBSP_CONFIG variable.
`shib-schemas`	Corresponds to SHIBSP_SCHEMAS variable.

An additional parameter, `server-name` (shown above), is used to explicitly signal the proper virtual hostname to use for the server instance loading the module.

To use a parameter, you supply it on the same line with `Init fn="nsapi_shib_init"` in the **magnus.conf** file as a `param="value"` pair.

Enabling the Module for Authentication

Getting the module integrated with the server is a bit convoluted, but the basic steps are:

- Enable the handler function (the thing that handles requests to "/Shibboleth.sso") for the appropriate internal MIME type and URL pattern.
- Enable the authentication function for the content you want to protect.

Handler Configuration

Part of the NSAPI module includes a "Service" function that responds to requests prefixed by the `handlerURL` established for the SP, typically some variant of "/Shibboleth.sso". There are a few configuration changes required to make this work properly, though some of the steps seem to be superfluous on some iPlanet/Sun versions. This combination of steps seems to be the most effective across versions.

First, create a custom MIME type for the ".sso" extension by adding a line to the **mime.types** file.

Add to mime.types

```
type=magnus-internal/shibboleth     exts=sso
```

Secondly, add the necessary "NameTrans" and "Service" commands in the "default" Object, and a custom Object type, in **obj.conf**. Normally these commands tend to be grouped together by command type and function in the file.

Add to obj.conf

```
<Object name="default">
...
NameTrans fn="assign-name" name="shib-handler" from="/Shibboleth.sso/*"
...
Service method="(GET|POST)" type="magnus-internal/shibboleth" fn="shib_handler"
...
</Object>

<Object name="shib-handler">
ObjectType fn="force-type" type="magnus-internal/shibboleth"
</Object>
```

Use of Sun Web Server 7.x

This version has broken some internal NSAPI behavior such that the system no longer accurately reports the port and scheme of the active request to the module. To work around this, `security_active` and `server_portnum` parameters can be used to supply these settings directly to the "Service" function. To use the physical values associated with the request you do (all on one line):

Alternate obj.conf Content for Sun Web Server 7.x

```
Service method="(GET|POST)" type="magnus-internal/shibboleth" fn="shib_handler" \
    security_active="$security" server_portnum="$env{'SERVER_PORT'}"
```

For virtual scenarios involving load balancers, you can explicitly control the port and security flag directly rather than relying on the macro variables.

After this work is complete, you should be able to verify success by accessing a handler directly, such as `https://hostname/Shibboleth.sso/Session`.

Authentication Configuration

To enable authentication, either actively or passively, you need to install an "AuthTrans" command in an appropriate location for the content you want to protect. One option is to install the command globally in a passive configuration in the "default" Object, and then use the `<RequestMapper>` for content-specific rules:

Add to obj.conf for Passive, Global Use

```
<Object name="default">
...
AuthTrans fn="nsapi_shib"
...
</Object>
```

Another option is to label content and provide an explicit command for that named Object:

Add to obj.conf for URL-based Authentication

```
<Object name="default">
...
NameTrans fn="assign-name" name="secure" from="*/secure/*"
...
</Object>

<Object name="secure">
AuthTrans fn="nsapi_shib" requireSession="1"
</Object>
```

Still another option is to do the same using a ppath-based Object:

Add to obj.conf for Directory-based Authentication

```
<Object ppath="D:/WebRoot/secure/*">
AuthTrans fn="nsapi_shib" requireSession="1"
</Object>
```

Use of Sun Web Server 7.x

As above for handler configuration, use of Sun Web Server 7.x will usually require passing additional parameters to the "AuthTrans" function to establish the port and scheme. The syntax for that is the same as noted earlier.

Authorization

The module does not provide for integration with the native access control features of the web server. Therefore you will need to either perform such work within your application, or rely on the `<RequestMapper>` and the XML-based Access Control plugin (or an alternative plugin to the SP).

Content Settings

The SP supports an extensible set of content settings, properties that control how it interacts with requests and enforces various requirements.

These settings can be controlled either by passing `settting="value"` parameters to the "AuthTrans" function in **obj.conf**, or by attaching properties using the `<RequestMapper>` mechanism in the SP configuration. By default, the SP ships with the "Native" RequestMapper plugin enabled. This plugin is what permits settings to be attached using either the native **obj.conf** syntax, or the XML syntax.

To use the native approach, you simply add a parameters with the name of the content setting and the value you want.

For example, to set `applicationId` to `foo`, the "AuthTrans" command would change to:

```
AuthTrans fn="nsapi_shib" applicationId="foo"
```

***obj.conf* Takes Precedence**

The native configuration will take precedence over anything you put into the `<RequestMapper>`, so be careful if you combine the two.

For more information about using the `<RequestMapper>` feature, refer to the NativeSPRequestMapHowTo topic.