Primetime Streaming Server 1.0

Getting Started Guide

Contents

Primetime Streaming Server

Adobe Primetime Streaming Server (PSS) is an Adobe Primetime Platform component that includes the capabilities of both Primetime Packager and Primetime Origin Server.

You can configure PSS to assume the role of a packager and an origin server simultaneously. Alternatively, configure PSS to assume either of the two roles.

Deploying Primetime Streaming Server

PSS comes as a deployable package that comprises a zipped jar file `pss.jar`.

To deploy the package, extract the content of the zip file on your computer. The deployment directory contains the configuration folders for both the Packager and Origin components.

You can launch either or both the components during startup, based on the configuration parameters you specify in the `pss.xml` file located in the conf folder.

The lib folder contains the external dependencies. Log files are created within the log directory. The cred directory contains the default PHDS/PHLS certificates.

The packager_conf folder in the installation directory contains the configuration files for the Packager. The origin_conf folder contains the configuration files for the Origin component.

Depending on your operating system, use the launch script for Windows or Linux to launch PSS.

Here is a sample pss.xml file with configuration parameters for the Packager and Origin components:

```
pss_install/
    conf/ <- PSS configuration folder
          pss.xml
       packager_conf/ <- Packager's configuration folder
         packager.xml
           containers/
              container_1/
                 container.xml
                    streams/
                         stream_1/
                              stream.xml
                           stream_2/
                             stream.xml
       origin_conf/ <- Origin's configuration folder
          server.xml
            containers/
               container_1/
                   container.xml
                       streams/
                          stream_1/
                            stream.xml
                          stream_2/
                            stream.xml
```

```
        logs/
          pss_0.log
             http/
               access.0.log
       creds/
              sd/...
               static/...
         webroot/
```

The format of configuration files for the Packager and Origin components is similar except for the differences in the Configuration section.

The pss.xml file in the conf folder contains PSS-specific configuration parameters.

To enable PSS to package and host a stream, configure the corresponding `stream.xml` file for both Packaging and Origin components.

The Packaging component can push the packaged stream either to the Local Origin or to other HTTP targets.

Note: Normally, Primetime Packager comes with a bundled lightweight Local Origin. However, the Packaging component of PSS does not contain this Local Origin. PSS contains a single Origin component whose content is stored on disk inside the webroot folder. The configuration parameters for the Origin component are defined in the origin_conf folder. This Origin component includes the capabilities offered by the standalone Primetime Origin. It can accept input from a local Packaging component, remote packager, or an encoder.

PSS Configuration

The configuration file hierarchy and format for PSS is the same as those for standalone Packager and Origin components. You can deploy configuration files from a standalone component on PSS, except for these limitations:

- Lightweight local origin that is bundled with a standalone Packager is inactive. The configuration for the lightweight local origin is specified in the file Packager_Install > conf > http > origin.xml. This file is not present in the packager_conf folder inside the PSS. Remove this file if you copy configuration parameters from the conf folder of a standalone Packager to the packager_conf folder of PSS.
- The standalone Packager has an option to publish packaged content to local disk through the LocalDisk target (config is //OutputPipeline/Output/LocalDisk in stream.xml) . Because the Origin component runs with the Packager in PSS, ensure that the output directory does not come into conflict with the Origin's root folder while writing the content to disk through LocalDisk Output config. By default, webroot is the root folder of the Origin component of PSS. Therefore, the ContentPath variable inside //OutputPipeline/Output/LocalDisk config should not point to any path inside the webroot folder.
- Because Local Origin is not available in PSS, the LocalOrigin output option is not available in the `stream.xml` file for the Packager component of PSS.

- As per the specifications for Primetime Packager, you can use //OutputPipeline/Output/HttpPush configuration parameter to push the packaged content to a remote HTTP endpoint. To publish the packaged content to the Origin component for the same PSS deployment, use the //OutputPipeline/Output/HttpPush/TargetURL configuration parameter and specify localhost as the value for host.

Here is a sample `stream.xml` file that configures two HTTP push targets. The first HTTP push target pushes the content to a remote origin hosted at origin-server1:8090. In contrast, the second target pushes the content to the Origin component of the same PSS deployment at the path */_default_/_default_*.

```
<Stream>
    <Input>
        <TSMulticast>
            ...
        </TSMulticast>
    </Input>
    <OutputPipeline>
        <OutputType>HDS</OutputType>
        ....
        <Output>
            <HttpPush>
              <TargetURL>http://origin-server1:8090/_default_/_default_</TargetURL>
            </HttpPush>
            <HttpPush>
            <!-- This will publish content to Origin of the same PSS installation.-->
                <TargetURL>http://localhost:8090/_default_/_default_</TargetURL>
            </HttpPush>
            <LocalDisk>
                <!-- Write fragments to some disk location . -->
                <ContentPath>packager_local_out/HDS_1</ContentPath>
            </LocalDisk>
        </Output>
    </OutputPipeline>
</Stream>
```

Configuration parameters for Packager component

PSS includes the following configuration changes in the `stream.xml` file for the Packager component:

Configuration parameter	Description	Type
//OutputPipeline/Output/LocalDisk/ContentPath	Mandatory configuration parameter for Local file output path if LocalDisk output is specified. ContentPath of LocalDisk target should not overlap with webroot path of the Origin component.	String

The following configuration changes are introduced in the `pss.xml` file:

Configuration	Description	Type
//Config/OriginEnabled	Optional parameter that specifies whether the Origin component of PSS should be started. The default value is true.	String with two supported values: true / false
//Config/PackagerEnabled	Optional parameter that specifies whether the Packager component of PSS should be started. The default value is true.	String with two supported values: true / false

Here is a sample configuration for a pss.xml file:

```
<Config>
    <!-- Set this false to disable the Origin Component. -->
    <OriginEnabled>true</OriginEnabled>
    <!-- Set this false to disable the Packager Component. -->
    <PackagerEnabled>true</PackagerEnabled>
</Config>
```

JMX operations

PSS includes an additional key property in the Mbean Object Name to differentiate between the Packager and Origin modules while preserving the order of the existing property list. To maintain the MBean tree view in JConsole, the new key appears at the beginning of the property list called group. The value of the property group is either Packager or Origin.

For PSS, the Object Name for event1 MBean is the following:

Object Name for event1 MBean in PSS	com.adobe.fms:group=Packager,type=Stream,name=event1,parentDomain=com.adobe.fms,parentType=StreamContainer,parentName=channel1,component=default

JConsole groups the MBeans by type. Stream Module MBeans for Origin and Packager are grouped under the type Stream. Subsequently, the tree is bifurcated in two nodes, one each for Packager and Origin.

Similarly, Stream Container MBeans are grouped together. Thereafter, they are bifurcated by Origin and Packager nodes.

The following is a sample PSS MBean tree structure in JConsole:

```
com.adobe.fms
 - Stream <- This node contains all the Stream Modules from both Origin and Packager
          -Packager <- This node will contain all Stream Modules from Packager
               -Packager Stream Module MBeans
          -Origin <- This node will contain all Stream Modules from Origin
                            Origin Stream Module MBeans
 - StreamContainer <- This node contains all the Stream Container from both Origin and
Packager
          -Packager <- This node will contain all Stream Container from Packager
               -Packager Stream Container MBeans
          -Origin <- This node will contain all Stream Container from Origin
                            Origin Stream Container MBeans
```

The remaining hierarchy for Packager MBeans and Origin MBeans is same as the hierarchy that appears Live Packager's or Origin's JConsole view.

To provide the consistency with clients, such as OSS, the same Object Names for the Packager and Origin MBean are used in PSS. The following is a sample of the entire MBean Tree in JConsole for PSS:

```
com.adobe.fms
         - OriginServer     <- Origin Server MBean
         - Server           <- Packager MBean
         - Stream           <- All Stream Module MBeans
            -Packager
                   -Packager Stream Module MBeans
            -Origin
                   -Origin Stream Module MBeans

          - StreamContainer <- All Stream Container MBeans
            -Packager
                  -Packager Stream Container MBeans
            -Origin
                  -Origin Stream Container MBeans
```

Logging

PSS uses java.util.logging for logging. You can configure logging using the properties file `logging.properties` in the install directory. The scripts `pss_start.bat` and `pss_start.sh` `load` the `logging.properties` file from the install directory.

The default file output appears in the logs directory under the install directory. By default, the `logging.properties` file is configured to generate PSS logs in the `pss.<n>.log` file where n is the log file number. Logs for the Packager component are generated in the `packager.log` file. In contrast, logs for the Origin component are generated in the `origin.log` file.

Access logs for the Origin component are generated in the http folder under the logs folder.

Note: The `pss.log` file contains the combined logs for both the Packager and Origin component.

You can modify the default `logging.properties` file to remove the `packager.log` and `origin.log` files. The `pss.log` file suffices because it has all the consolidated logs.

The following table explains the domains used for logging:

Logging Namespace

Name	Description
com.adobe.fms	The root namespace for all PSS logging. Output is set to ./logs/pss.<n>.log by default
com.adobe.fms.http	Used for access logs of the Origin Server. Will be logged in ./logs/http/access.log by default.
com.adobe.fms.Origin.StreamContainer.<container_name>	Logging namespace for Origin's Stream Container named "container_name". Will be logged in ./logs/origin.log by default
com.adobe.fms.Origin.StreamContainer.<container_name>.Stream.<stream_name>	Logging namespace for Origin's Stream module named "stream_name". Will be logged in ./logs/origin.log by default
com.adobe.fms.Packager.StreamContainer.<container_name>	Logging namespace for Packager's Stream Container named "container_name". Will be logged in ./logs/packager.log by default
com.adobe.fms.Packager.StreamContainer.<container_name>.Stream.<stream_name>	Logging namespace for Packager's Stream module named "stream_name". Will be logged in ./logs/packager.log by default

Similar to the logging scheme for standalone Packager and Origin, the logging output for PSS streams are prefixed with **[<component name>, containerName,streamName]**. In addition, the logging output for PSS containers are prefixed with **[<component name>, containerName].** The component name for the Packager component is Packager. Similarly, the component name for the Origin component is Origin. Prefixes are used to distinguish the packager and origin stream logs in the combined `pss.log` file.

Packager Auth Plugin Interface

Primetime Live Packager implements a default authentication scheme. However, you can implement your custom logic and place the JAR file for the implementation in the root directory.

Here are the steps to add a custom authentication plugin:

1. Implement the following interface with the custom authentication scheme (in function authenticateRequest):

```
public interface AuthAdaptor
```

```
{
    public interface Status
    {
        public boolean getStatus();
        public String getReason();
    }

    /**
     * Hook to initialize the plug-in, passing the &lt;init&gt; element within
     * the AuthAdaptor config element or <tt>null</tt> if no initialization
     * configuration was provided.
     * This method is invoked during module initialization and is guaranteed
     * to have finished execution before any calls to {@link #authenticateRequest}are made.

     * @param configXML The &lt;init&gt; child element of the plug-in config
     * element.
     * @param logger Logger to be used for logging
     */
    public void initialize(Element configXML, Logger logger)
    throws Exception;

    /**
     * Hook to check if authentication is to be done for the request, passing the
&lt;CmdType&gt;
     * This method is invoked during CONNECT.
     *
     * @param cmdType The &lt;CmdType&gt;. At present only authentication on CONNECT is
supported.
     * element.
     * @param url Input URL
     */
    public Status isAuthEnabled(CmdType cmdType, String url)
    throws Exception;

    /**
     * Hook to authenticate the request, passing the &lt;CmdType&gt;
     * This method is invoked during CONNECT.
     *
     * @param cmdType The &lt;CmdType&gt;. At present only authentication on CONNECT is
supported.
     * element.
     * @param url Input URL
     */
    public Status authenticateRequest(CmdType cmdType, String url)
    throws Exception;

    /**
     * Hook to save raw credentials.
     * This method is invoked during container creation.
     *
     * @param filePath The absolute &lt;directory path&gt; of the out file. FileName is to be
appended by the implementor
     * @param credentials Map of username and passwords to be saved.
     */
    public Status saveRawCredentials(String filePath, Map<String, String> credentials)
    throws Exception;

    /**
     * Hook to save encoded credentials.
     * This method is invoked during container creation.
     *
     * @param filePath The absolute &lt;directory path&gt; of the out file. FileName is to be
appended by the implementor
     * @param credentials list of encoded credentials to be saved.
     * The string would be assumed to be encoded and would be saved as it is.
     */
    public Status saveEncCredentials(String filePath, List<String> encodedCredentials)
    throws Exception;

    enum CmdType
    {
        CONNECT
    }
}
```

2. Placed in JAR file that you build at the following the directory *<ROOT DIR>/conf/rtmp/lib* for Live Packager and the directory *<ROOT DIR>/packager_conf/rtmp/lib* for Primetime Streaming Server.
3. Add the following configuration to the RTMP configuration file:

```
LivePackager: <ROOT DIR>/conf/rtmp/server.xml:
<Root Dir>
<Config>
...
   <AuthPlugin>
    <ClassName>com.adobe.fms.util.authadaptor.AuthAdaptorPlugin</ClassName>
    <Init>
        <ConfRoot>./packager_conf</ConfRoot>
        <UserFile>users.dat</UserFile>
    </Init>
   </AuthPlugin>
</Config>

Primetime Streaming Services: <ROOT DIR>/packager_conf/rtmp/server.xml:
<Root Dir>
<Config>
...
   <AuthPlugin>
    <ClassName>com.adobe.fms.util.authadaptor.AuthAdaptorPlugin</ClassName>
    <Init>
        <ConfRoot>./conf</ConfRoot>
        <UserFile>users.dat</UserFile>
    </Init>
   </AuthPlugin>
</Config>
```

The following table describes the configurations elements for RTMP authentication:

Name	Description
AuthPlugin	Tag to be added to enable authentication.
ClassName	Name of the main class of the plugin.
Init	Custom element where you specify initialization configuration for the auth plugin. The following is the initialization configuration for the

	default implementation: • ConfRoot: Path to the configuration directory • UserFile: Name of the users file (where user names and passwords are saved)

4. Restart Live Packager.
   Note: If the custom authentication plugin doesn't use the default `users.dat` format (for on-premise cases), use a custom tool to save the credentials. In a hosted scenario, provide an implementation of `saveCredentials` in the auth plugin.