# GI – MAS

## Content



History

| Version | Date | Adjustments |
|---|---|---|
| A | 2006-10-16 | First version. |
| B | 2006-12-07 | Added Stream building prerequisites. |
| C | 2008-08-05 | Updated patch building instructions. |

# 1 Introduction

This document describes how to build the MAS and how to setup the build environment on solaris10.

It will deal with building complete versions from LATEST, building specific builds , building patches and emergency corrections. It also shows how to build the application and media content package builder.

# 2 Generation Information

## 2.1 Environment

It is assumed that Java 5 and Ant is installed in user/local as described below for building on Solaris otherwise the directories used in the environment settings below must be changed to support the build environment.

The following environment must be set before any build action can take place:

```
setenv JAVA_HOME /usr/local/jdk1.5.0_05
setenv ANT_HOME /usr/local/ant-1.6.5
setenv COBERTURA_HOME /vobs/ipms/mas/tools/cobertura
setenv PATH ${HOME}/bin:${JAVA_HOME}/bin:${ANT_HOME}/bin:${PATH}
```

## 2.2 Stream Make Preconditions

It is important that the object and library files of Stream (C++) are properly cleaned/deleted before build is commenced. It has been show more than once that the make files has failed to fulfill this. Hence, one precondition of generating MAS is to ensure that Stream is properly cleaned.

The simplest way to ensure that Stream C++ is clean is to let ClearCase check for view private files and if there are remove them:

```
rm –rf `ct lsprivate`
```

NOTE. This will remove every view private file in the entire view so you have to be sure of that you don't have view private files that you want to keep. But this should not be of any problem since you have a specific view dedicated for builds.

## 2.3 Building a complete version from LATEST

The following actions must be made to build a complete and labeled MAS version. The label can be anything as long as it selects a consistent MAS.

1. Log in to the build machine.
2. Create or use a view that selects the labeled version in clearcase.
3. Start the view and cd to **/vobs/ipms/mas/tools/svcs_tools/**.
4. Start scripting to save all output from the build.

   **script build.tmp**

5. Run mkdeliv_pkg.sh giving it the name of the version you are about to build. For this example we want to build a preliminary first version of the **MAS**, i.e. **MAS_P1A**.

   **./mkdeliv_pkg.sh MAS_P1A**

6. The script will suggests a label based on previous build numbers. Select that label and write down the label for later usage. For this example lets

assume that the label was **MAS_P1A.154**. **NOTE** ! If you break, ctrl-c, the build process the script will not be able to restore your configuration specification.

7. The script will now label the entire MAS vob with **MAS_P1A.154**.
8. The script will then save your configuration specification to be able to restore it later, and change it to only select files with the **MAS_P1A.154** label.
9. The script will then build the **MAS** and create a solaris package.
10. The delivery file will be created in the **/vobs/ipms/mas** directory and named **mas_<version>.mas0001.solaris10.tar.gz**.
11. Exit the scripting started at bullet 4.

    **exit**

12. Open build.tmp on an editor, vi, emacs or xemacs and go through the output to verify that no errors occurred during compilation or solaris package build.
13. Finished !

## 2.4 Building a complete version from LABEL

The following actions must be made to build a complete and labeled MAS version. The label can be anything as long as it selects a consistent MAS.

1. Log in to the build machine.
2. Create or use a view that selects the labeled version in clearcase.
3. Start the view and cd to **/vobs/ipms/mas/tools/svcs_tools/**.
4. Start scripting to save all output from the build.

   **script build.tmp**

5. Run mkdeliv_pkg.sh giving it the label for the version and build you are about to build. For this example we want to build a preliminary first version build 154 of the **MAS**, i.e. **MAS_P1A.154**.

   **./mkdeliv_pkg.sh MAS_P1A.154 –l**

   **NOTE !** The switch after the label is the small letter L.

6. The script will ask if you are sure.
7. The script will then save your configuration specification to be able to restore it later, and change it to only select files with the **MAS_P1A.154** label. **NOTE** ! If you break, ctrl-c, the build process the script will not be able to restore your configuration specification.
8. The script will then build the **MAS** and create a solaris package.

9. The delivery file will be created in the **/vobs/ipms/mas** directory and named **mas_<version>.mas0001.solaris10.tar.gz**.
10. Exit the scripting started at bullet 4.

    **exit**

11. Open build.tmp on an editor, vi, emacs or xemacs and go through the output to verify that no errors occurred during compilation or solaris package build.
12. Finished !

## 2.5 Building an Application and Media Content package builder

The following actions must be made to build a complete and labeled Application and Media Content Package builder version.

1. Create or use a view that selects the labeled version in clearcase.
2. Start the view and cd to **/vobs/ipms/mas/amcpm**.
3. Start scripting to save all output from the build.

   **# script build.tmp**

4. Run mkdeliv.sh giving it the name of the version you are about to build.
   **# ./mkdeliv.sh**
5. Exit the scripting.
   **#exit**

6. Inspect build.tmp for any errors that caused the build to fail. If not the tar file is located in **/vobs/ipms/mas/amcpm** with the name indicated by the build script.

## 2.6 Building a patch

### 2.6.1 Prepare patch specific files

The directory /vobs/ipms/mas/patches/ contain files specific for this particular patch build. The following files may need to be updated (depending on the needs for the particular patch to build.):

- patches/patch.cfg: Contains the configuration for this patch including the patch name and build-number. This file must always be updated. See the comments in the file for instruction on how to fill it in.

- patches/patch_files.lst: This file contain a list of all MAS-files that shall be patched, i.e. all files that have been changed compared to the base version of the component. It is very important that all changed files are listed here otherwise they will not be included in the patch.

- patches/comp_checkinstall: This is a script run before patch installation in order to check prerequisites. It for example checks for correct OS version. This file can usually be left untouched.

- patches/comp_checkremove: This is a script run before patch removal in order to check prerequisites. Currently does nothing. It can usually be left untouched.

- patches/comp_preinstall: Script run prior to patch installation. Stops the MAS if not already stopped. It can usually be left untouched.

- patches/comp_postinstall: Script run after patch installation. This script is responsible for updating any configuration files that need this. It will register the new version in MCR and it will start the MAS again (if it was started prior to patch installation). This script often need to be modified. It includes functions to update configuration files by using an XSLT-transform. Modify the in-line XSLT code in order to adapt to the config changes needed or comment-out the usage of these functions if no config needs to be updated.

- patches/comp_preremove: Script run prior to patch removal. It saves the start/stop state of the MAS and it stops the MAS if not already stopped. It can usually be left untouched.

- patches/comp_postremove: Script run after patch removal. This script is responsible for restoring any configuration changes made during the patch installation. It will also restore the version registered in MCR and it will start the MAS again (if it was started prior to patch removal). This script often need to be modified. It includes functions to restore configuration files by using an XSLT-transform. Modify the in-line XSLT code in order to adapt to the config changes needed or comment-out the usage of these functions if no config needs to be restored.

Note: There also exist a script, *xmlmodify.sh*, that can be used to modify the configuration files in the comp_postinstall and comp_postremove scripts. It can replace the use of XSLT-transforms for most cases and is simpler to use if one is not familiar with XSLT-transforms. The script can be found under *mas/tools/svcs_tools/*.

Please also refer to [1] "DG – Package patch" for more detailed information on patch packages.

### 2.6.2 Private patch build (Test build)

1. Preferably use a clean build view with no checked-out or view private files.

2. Manually build all parts of MAS that have been changed in this patch or, if only jar-files have been changed, leave the build to the patch build script (not well tested though). If unsure it is recommended to perform a full MAS build. One way to accomplish this is to perform normal build according to 2.3 but adding the arguments "-p –noclean". This will create a private MAS build without cleaning up jar-files and lib-files afterwards. These files will then be picked up by the patch build script.

3. Go to /vobs/ipms/mas/tools/mk_patch directory.

4. If all patch files have been manually built already, perform:
   **./mkpatch_pkg.sh -p –nojarbuild**
   in order to perform a private build of the patch.
   If only jar-files need to be patched and they have not been manually built, perform:
   **./mkpatch_pkg.sh –p**
   which automatically builds necessary jar-files and then creates the patch.

5. Verify that no errors occured during patch build.

6. The resulting patch files can be found in the directory:
   /vobs/ipms/mas/MOBYmas_patch/

   The patch file will be named **<PatchId>.<Build>.mas0001.solaris10.zip** and a corresponding readme template will be named:
   **README_<PatchId>.<Build>.mas0001.solaris10.template**

7. Edit the created README template and modify the extension to ".txt" instead of ".template".

8. The patch is ready to be basic tested.

### 2.6.3 Release patch build (Labeled build)

1. Use a clean build view with no checked-out or view private files.

2. Manually build all parts of MAS that have been changed in this patch or, if only jar-files have been changed, leave the build to the patch build script (not well tested though). If unsure it is recommended to perform a full MAS build. One way to accomplish this is to perform normal build according to 2.3 but adding the arguments "-p –noclean". This will create a private MAS build without cleaning up jar-files and lib-files afterwards. These files will then be picked up by the patch build script.

3. Go to /vobs/ipms/mas/tools/mk_patch directory.

4. If all patch files have been manually built already, perform:
   **./mkpatch_pkg.sh –nojarbuild**
   in order to perform a private build of the patch.
   If only jar-files need to be patched and they have not been manually built, perform:
   **./mkpatch_pkg.sh**
   which automatically builds necessary jar-files and then creates the patch.

5. Verify that no errors occurred during patch build.

6. When that patch is built, a directory will be created under /**vobs**/**ipms**/**mas**/**patches**/ directory, named after the patch, which will contain the patch zip-file together with some input files for the patch-build.

7. Verify that the patch contain all files needed and that all binaries are the correct version.

8. A template for the README file will be automatically created under the **/vobs/ipms/mas/MOBYmas_patch/** directory. Edit this created README template to include all patch/customer specific information. Modify the extension to ".txt" instead of ".template" and move the ready readme-file to /**vobs**/**ipms**/**mas**/**patches**/**<Patch name>**/

9. Check in the /**vobs**/**ipms**/**mas**/**patches**/**<Patch name>**/ directory including the contained files.

10. The patch is ready to be delivered to component tests. Copy the zip-file and the readme-file to the delivery area.

# 3 References

**[1]** DG Package patch
1.DG.CRH 109 581.1

# 4 Terminology

| | |
|---|---|
| MAS | Media Access Server |