



GMPS Guide

Version 5.0

Windows, Unix



SASO
Software & Consulting

GMPS Guide

Version 5.0 and later

Windows, Unix Edition



GMPS Guide

Document Number 9000-005

July 2009

SASO, Software & Consulting GmbH, Grünwalder Straße 1, D-81547 München

Important Notice

Copyright 2009 SASO, Software & Consulting GmbH, All rights reserved.

Permitted Usage

This document is protected by copyright and contains information proprietary to SASO. Any copying, adaption, distribution or public display of this document without the express written consent of SASO is strictly prohibited. The receipt or possession of this document does not convey any right to reproduce or distribute its contents or to manufacture, use or sell anything that it may describe, in whole or in part, without the specific written consent of SASO.

Trademarks

SASO, SASO, Software & Consulting GmbH, the SASO logo, **GMPS**, the **GMPS** logo among others are trademarks or registered trademarks of SASO, Software & Consulting GmbH in Germany. All other names are used for identification purposes only, and are trademarks or registered trademarks of their respective companies.

Sun, Solaris and Java are trademarks or registered trademarks of Sun Microsystems, Inc.

Microsoft, the Microsoft logo, the Microsoft Internet Explorer logo, Windows, the Windows logo, Windows NT, the Windows Start logo are trademarks or registered trademarks of Microsoft Corporation in the United States and other countries.

Subversion, the Subversion logo are trademarks or registered trademarks of The Subversion Corporation.

ClearCase, the ClearCase logo are trademarks or registered trademarks of the International Business Machines Corp., registered in many jurisdictions worldwide.

Warranty Disclaimer

This document and its associated software may be used as stated in the underlying license agreement. SASO, Software & Consulting GmbH expressly disclaims all other warranties, express or implied, with respect to the media and software product and its documentation, including without limitation, the warranties of merchantability or fitness for a particular purpose or arising from a course of dealing, usage or trade practice.

Contents

Content

Preface.....	6
About this manual	6
Typographical Conventions	7
Technical Support.....	8
1. Introduction.....	9
1.1 Purpose of GMPS 5.0.....	9
1.2 Features of GMPS 5.0.....	9
2. Installing GMPS.....	10
3. Setting up the demo projects.....	10
3.1 Installation of GMPS-Demo package.....	10
3.2 Content of the GMPS-Demo package	11
3.3 Important commands to get an overview.....	11
3.4 Important configuration files of the demo projects.....	12
4. Configuring GMPS	12
4.1 Principle project structures and build philosophy	12
4.2 Environment variables.....	13
4.3 Structure of gmps-home-dir (documentation not ready jet).....	13
4.4 Configuration files	13
4.4.1 gms_config.xml.....	14
4.4.2 tools.xml	15
4.4.3 target.xml	15
4.4.4 SetEnv.xml	16
4.4.5 gms_targets.xml	17
4.4.6 gms_magic.xml	17
4.4.7 SetMacro.xml	19
4.4.8 gms_project_vobs.xml	19
4.4.9 template_section.xml.....	19
4.4.10 target.options.....	19
5. Main programs of GMPS	20
5.1 gms_make	20
5.1.1 Synopsis Command line.....	20
5.1.2 Workflow of gms_make	22

5.2	gms_register.....	22
5.2.1	Synopsis command line.....	23
5.2.3	Workflow of gms_register.....	23
5.3	gms_unregister.....	24
5.3.1	Synopsis command line.....	24
6.	Some more GMPS tools.....	25
6.1	gms_list.....	25
6.2	gms_list_dep.....	26
6.3	csproj2cfg (does not work in Demo-Version).....	26
6.4	netparser (does not work jet).....	27
6.5	dvp2cfg (does not work jet).....	27
6.6	gms_mkproj (does not work jet).....	27
7.	GMPS and XSL-Templates.....	27
7.1	Define new templates.....	27
7.1.1	Modify target.xml.....	27
7.1.2	Modify tools.xml.....	28
7.1.3	Template for editing tools.xml (documentation is not ready jet).....	28
7.1.4	Write xsl-templates.....	28
7.1.5	Edit the file template_section.xml.....	29
8.	Index of xml-tags for the GMPS Target configuration file.....	29
8.1	C/C++ projects.....	29
8.2	JAVA projects.....	32
8.3	C# projects.....	33
8.4	Miscellaneous.....	34
9.	Appendix.....	35
9.1	GMPS and xsltproc.....	35
9.2	GMPS and omake.....	35
9.3	GMPS and platforms.....	35
9.4	External tools.....	35

Preface

GMPS is a toolkit used to setup and maintain reliable production and release processes in conjunction with Subversion or ClearCase.

About this manual

This manual is for users who are new with **GMPS** and who may have other experiences with make tools. It provides information on setting up **GMPS**, gives an overview of building programs using **GMPS**, and describes **GMPS** usage. The appendices contain information on error messages, macros and rules, compatibility, and regular expressions.

If you are already familiar with the build process and Makefiles, the recommended sequence for proceeding through the documentation is:

- Read Chapter 1, *Introduction*, for general information about GMPS
- Read Chapter 2, *Installing GMPS*
- Read Chapter 4, *Configuring GMPS*
- Read the other chapter as required

If you are not familiar with the build process and Makefiles, the recommended sequence for proceeding through the documentation is:

- Read Chapter 1, *Introduction*, for general information about GMPS
- Read Chapter 3, *Setting up the demo projects*
- Read Chapter 2, *Installing GMPS*
- Read Chapter 4, *Configuring GMPS*
- Read the other chapter as required

Typographical Conventions

This manual uses the following typographical conventions:

- *gmps-home-dir* represents the directory into which the GMPS Product Family has been installed. By default, this directory is /usr/saso on UNIX and C:\SASO\GMPS-Demo\gmps on Windows.
- **Bold** is used for names the user can enter; for example, all command names, file names, and branch names.
- *Italic* is used for variables, document titles, glossary terms, and emphasis.
- A monospaced font is used for examples. Where user input needs to be distinguished from program output, **bold** is used for user input.
- Nonprinting characters are in small caps and appear as follows: <EOF>, <NL>.
- Key names and key combinations are capitalized and appear as follows: SHIFT, CTRL+G.
- [] Brackets enclose optional items in format and syntax descriptions.
- { } Braces enclose a list from which you must choose an item in format and syntax descriptions.
- | A vertical bar separates items in a list of choices.
- ... In a syntax description, an ellipsis indicates you can repeat the preceding item or line one or more times. Otherwise, it can indicate omitted information.
- If a command or option name has a short form, a “medial dot” (•) character indicates the shortest legal abbreviation. For example:

lsc•heckout

This means that you can truncate the command name to **lsc** or any of its intermediate spellings (**lcs**, **lsche**, **lschec**, and so on).

Technical Support

If you have any problems with the software or documentation, please contact SASO Technical Support via telephone, fax or electronic mail as described below. For information regarding support hours, languages spoken, or other support information, click the Technical Support link on the SASO Web site at www.saso.de.

Your Location	Telephone	Facsimile	Electronic Mail
North America, Europe, Middle East, Africa, and Asia Pacific	+49-89-455-786-77	+49-89-455-786-79	support@saso.de

1. Introduction

1.1 Purpose of GMPS 5.0

GMPS is a toolkit used to setup and maintain reliable production and release processes in conjunction with Subversion or ClearCase. Using GMPS, developers only need to know some very basic things about the GMPS process to do their job. They may build anything from a single target to a complete release with one single command. Different build modes (e. g. debug) are supported. If new sources are added for a target, it's enough to modify a very simple configuration file to update the process. Target specific information (like a list of source names) is completely separated from common information.

Administrators profit from:

- a unified structure for the whole project,
- avoidance of redundant information,
- Separations of different aspects in a complex make system.

The close connection to Subversion or ClearCase guaranties that any change in the configuration could be turned back if necessary. The design of GMPS encourages the user to choose a directory or VOB structure with a clear concept that has evolved over time through the use in a variety of projects. GMPS is flexible enough to meet project specific requirements but it doesn't allow anything.

1.2 Features of GMPS 5.0

GMPS is currently used in several projects with various customizations. From time to time, general interest customizations are merged into a new GMPS version. The following list of features is just the basic subset of the generic version of GMPS:

- Mixed platform (e. g. SUN Solaris, RedHat Linux, Windows) support.
- Build mode selection.
- Local build options.
- Straight forward definition and selection of targets and phonies.
- Integrated maintenance of directory and VOB structures for registration and release processes.
- Detailed check-and-execution reports.
- Optimized release tag or label application for better performance.

The main scripts of GMPS are:

- `gms_make`, the central control program,
- `gms_register`, register a new target to the project specific target registry file "`gms_targets`" and update the associated `target.cfg` file,
- `gms_unregister`, unregister a target from the project specific target registry file "`gms_targets`" and update the dependent `target.cfg` file.
- `gms_prod`, produce a release, checkin the release and tag or label all associated sources. Check for consistency conflicts.

The master Makefile `gms_mastermake.mk` (see also workflow `gms_make`) is written in standard make language (trivial subset). `gms_make` uses XSLTproc to generate the Makefile from the defined

templates. All templates are written in XML-syntax, so the XSLTproc is required (included in the GMPS-package) and can handle all defined templates.

Normally the developer or build manager builds a target or release by calling the gms_make script. However, he/she can also work with the generated Makefile which resides in the associated project-directory. This Makefile can also be modified though it must be noted that the official production from the build manager works with an auto generated Makefile with the official defined flags, options and source dependencies.

2. Installing GMPS

This chapter describes how to install GMPS. You, the GMPS administrator, install GMPS by performing the following steps:

1. Install GMPS by clicking on gmps_setup.exe and then follow the instructions of the installation shield. The preset installation path is: C:\SASO\GMPS. You can change this path during the installation if you want to.
2. **a) For working with Subversion:** import all GMPS-files into a convenient repository in Subversion. Workflow as follows:

```
svnadmin create \sample_repository           – creates a new repository
cd C:\SASO\GMPS                             – change to installation path
svn import . file:///d:/sample_repository -m "Initial import" - import GMPS into subversion
svn list file:///d:/sample_repository       - check if GMPS is imported
```

- b) For working with ClearCase:** import all GMPS-files with clearfsimport. For example:
M:\admin_vu\vob1>clearfsimport C:\SASO\GMPS* .

3. Setting up the demo projects

3.1 Installation of GMPS-Demo package

The demo-version of GMPS 5.0 is only working with Windows.

1. Install GMPS by clicking on gmps_setup.exe and then follow the instructions of the installation shield. The preset installation path is: C:\SASO\GMPS-Demo. You can change this path during the installation if you want to.
2. **a) For working with Subversion:** import all GMPS-Demo files into a convenient repository in Subversion. Workflow as follows:

```
svnadmin create \sample_repository           – creates a new repository
cd C:\SASO\GMPS-Demo                       – change to installation path
```

```

svn import . file:///d:/sample_repository -m "Initial import" - import GMPS-Demo into
subversion
svn list file:///d:/sample_repository - check if GMPS-Demo is
imported

```

b) For working with ClearCase: import all GMPS-files with clearfsimport. For example:
M:\admin_vu\vob1>clearfsimport C:\SASO\GMPS* .

After the installation of GMPS you can start the program from your desktop or from the start-menu
-> programs -> GMPS 5.0. Starting GMPS the environment variables are set up automatically. Please
read instructions appearing in prompt command, maybe you have to change the GMPS_TOP path.

3.2 Content of the GMPS-Demo package

Structure of installation directory:

Content	Description
gmps	GMPS main program; for details read chapter 4.3 Structure of gmps-home-dir
sample_src	Includes the source files of the demo projects
sample_do	Directory for the derived objects (do's not in Version Control)
sample_release	Directory for the released objects (do's in Version Control)
sample_tools	Directory for the used tools (e.g. JDK, Open Watcom, etc.)
tags	Directory for tags (just for Subversion)
Uninstal.exe	Uninstall program for GMPS

3.3 Important commands to get an overview

For your first steps with GMPS and the demo projects we recommend to get an overview with the following commands (for all GMPS commands read chapter 5 and 6):

Command	Output
gms_list	show all registered targets of the project
gms_list_dep all	show graphically all dependencies between the registered targets. In this case "all" is the name of the phony (for phony read chapter 4.3.6)
gms_list_dep -man	show all options for gms_list_dep
gms_make -man	show all options for gms_make
gms_make excarea.exe	generate the Makefile and compiles the target in default build mode (debug). The generated Executable is now in \$GMPS_TOP/sample_do/bin/debug

<code>gms_make excarea.exe -release</code>	generate the Makefile and compiles the target in build mode release. The generated Executable is now in \$GMPS_TOP/sample_do/bin/release.
<code>gms_make excarea.exe -clean</code>	delete all generated files (do's) of this target. Not the released files!
<code>gms_make excarea.exe -gf -B</code> <code>gms_make java_pkg</code>	unconditionally make of all targets. build the phony "java_pkg", includes the targets "hallo.jar" and "KontoTest.jar"
<code>gms_make all -tag REL_VER_1.0</code>	compile all registered targets and create a tag in the repository/tags/REL_VER_1.0 with the content of "\$GMPS_TOP/*directories"
<code>gms_unregister hallo.jar</code>	unregister the target "hallo.jar". Call "gms_list_dep all" to see, that the target is deleted.
<code>cd.. (change directory to hallo-source directory)</code> <code>gms_register -f hallo.xml</code>	register the target "hallo.jar". Call "gms_list_dep all" to see, that the target is registered.

3.4 Important configuration files of the demo projects

The central configuration of the flags and tools you can find in tools.xml, abstract path: GMPS_TOP -> GMPSHOME -> etc -> GMPS_PROJECT -> GMPS_PLATFORM -> tools -> tools.xml

Path for sample-project: C:\SASO\GMPS-Demo\gmps\etc\sample\win32\tools\tools.xml

The individual configuration for sources and flags you can find for each target in the target.xml file

e.g. path for excarea.exe in sample-project: C:\SASO\GMPS-Demo\sample_src\cppexamp\excarea\excarea.xml

4. Configuring GMPS

4.1 Principle project structures and build philosophy

The separation of sources, derived objects (do's), used tools and releases is an essential principle of the build philosophy of GMPS.

That means that there are always at least four directories or VOBs (versioned object bases). For example:

.../src – for the sources

.../do – for the derived objects

.../tools – for the used tools

.../release – for the releases

- The directory structure of the src- and do-directories or -VOBs is the same or a mirror.
- All derived objects are stored into the do-directory or -VOB.
- The developer uses the pre-produced products as shared products ("wink in" in ClearCase)

- The developer may produce his/her own private products and overwrite the pre-produced products.
- The Makefile is stored and executed in the do-directory or -VOB. A Makefile is produced for every target.
- The tools-directory or -VOB contains all important tools (compiler, linker, etc.)

4.2 Environment variables

GMPS has the following environment variables, which need to be defined before you can start working with GMPS.

Environment variable	Purpose / Set up
GMPSHOME	The relative path to the GMPS-directory
GMPS_TOP	The absolute path to the project-directory
GMPS_PROJECT	The name of the project
GMPS_PLATFORM	You can choose between "win32" or "linux"
GMPS_BUILDMODE	Select "debug" or "release"
PATH	Add the path to the GMPS-Scripts to this variable
TMPS	Add the path for the temporary files

For Windows: To define all these environment variables you need to set them up in the control panel, menu item "system".

If you don't add a drive letter for GMPS_TOP, xsltproc don't reports an error in the XSL and XML tmp-files. Also make and omake are working without error, but in case of using ClearCase gms_make reports the error "GMPS_TOP is not defined". So you have to define the drive letter "GMPS_TOP:Z".

If the environment variable *TMP* is not set, gms_make takes this variable from SetEnv.xml or sets `TMP = $GMPS_TOP/$GMPSHOME/tmp` (just in Subversion).

4.3 Structure of gmps-home-dir (documentation not ready jet)

4.4 Configuration files

The following configuration files need to be distinguished:

- 4.4.1 gms_config.xml
- 4.4.2 tools.xml
- 4.4.3 target.xml
- 4.4.4 SetEnv.xml
- 4.4.5 gms_target.xml
- 4.4.6 gms_magic.xml
- 4.4.7 SetMacro.xml

- 4.4.8 gms_project_vobs.xml
- 4.4.9 template_section.xml
- 4.4.10 target.options

4.4.1 gms_config.xml

The gms_config.xml file is the central configuration file for a project. It contains all global defined tool paths, source paths, do paths release paths, all defined templates and some global settings. It also contains defined suffixes, buildmodes and extensions for built targets.

The gms_make, gms_register and gms_unregister scripts work with the gms_config.xml file.

Here the description of the included tags

<GENERAL_SETTINGS> contains general information about the project.

<VARIABLE> describes the environment variables.

<LOCATION> contains important information for gms_make. Structure as follows

```
<LOCATION
  SourceLocation = " "
  DoRoot = " "
  ReleaseRoot = " "
  BuildModes = " "
  DefaultBuildMode = " "
  Platforms = " "
/>
```

SourceLocation is the root-directory of the sources, relative path to \$GMPS_TOP.

DoRoot is the root-directory of the derived objects (do's), relative path to \$GMPS_TOP.

ReleaseRoot is the root-directory of the released objects, relative path to \$GMPS_TOP.

BuildModes contains potential buildmodes (release, debug) separated by space character. This attribute is used by gms_register for the registration of the targets.

DefaultBuildMode is used if there is no \$GMPS_BUILDMODE environment variable defined.

Platform contains potential platforms (win32, linux, etc.) separated by space character. This attribute is used by gms_register for the registration of the targets.

<DoDirPathExtensions Type> is used by gms_register. It describes the additional directories which are created in the do-directory. The attribute "Type" shows the targets and the attribute "DirExt" shows the directories.

Structure of <DoDirPathExtensions>:

```
<DoDirPathExtensions
  Type = " "
  DirExt = " "
/>
```

<VERSION_CONTROL_SYSTEM Name = "ClearCase or SVN" />

Sets up the version control system. Supported systems are ClearCase or Subversion (SVN).

<TARGET> describes the output-directory for targets, dependent of the type. That means the target is written in \$GMPS_TOP\DoRoot\@do_out\platform\buildmode (for do's) or/and \$GMPS_TOP\ReleaseRoot\@release_out\platform\buildmode (for releases). You have to define a target for each type.

Structure of <TARGET>:

```
<TARGET
  type = "bat" do_out = "bin" release_out = "bin"
/>
```

<EXCLUDEFLAGS> describes the flags which must not be used in the project. These flags are valid for all tools, there is no exception. If flags from this list appear in the Makefile, they will be deleted. The flags in the list must be written exactly, the first character can be "-" or "/".

Structure of <EXCLUDEFLAGS>:

```
<EXCLUDEFLAGS
  flags = '-YX -Fp -Yc -Fo -Yustdafx.h -Yu"stdafx.h" -FD -Fd'
/>
```

4.4.2 tools.xml

In tools.xml are defined all tools which are used in the project. tools.xml has several <TOOL> tag describing the individual tools.

Structure of <TOOL> tag:

```
<TOOL
  WIN32_PATH = " "
  LINUX_PATH = " "
  GLOBAL_FLAGS = " "
  RELEASE_FLAGS = " "
  DEBUG_FLAGS = " "
  WIN32_GLOBAL_FLAGS = " "
  WIN32_RELEASE_FLAGS = " "
  WIN32_DEBUG_FLAGS = " "
  LINUX_GLOBAL_FLAGS = " "
  LINUX_RELEASE_FLAGS = " "
  LINUX_DEBUG_FLAGS = " "
/>
```

The arguments are the same at all targets. Arguments that you don't need can be left out. The name of the tag has to be capitalized. The flags need to be in single quote signs and have to be separated by space character.

4.4.3 target.xml

For each target you have to define a target.xml-file. This file has the following sections:

- target description (type, name)
- sources and flags for single tools
- build process description (build tag)

It is necessary to define <TARGET> in the tag <TARGET> as follows:

```
<TARGET
  Name = "Name"
  Type = "dll | exe | jar | etc. | own type"
/>
```

With this name the target is registered and used in the whole project. The target name is case-sensitive.

The sources and flags which are edited with the same tool are united in one tag. Like in tools.xml all possible attributes for the two platforms (win32, linux) are defined here. The attributes that you don't need can be left out. The value of the attributes has to be in quote signs, sources and flags are separated by space character. At the end of the tag don't forget ">".

The build tag consists of the name and optional of the attributes for the sources. For example:

```
<EXECUTABLE/>
or
<DII
  DefSource = " "
/>
```

To avoid any conflicts the names of the tags are written capitalized (e.g. CPPSOURCES) in tools.xml and in target.xml in intercap (e.g. CppSources).

4.4.4 SetEnv.xml

In the file SetEnv.xml are defined the environment variables.

Structure of SetEnv.xml:

```
<SET_ENV>
  <Path
    Path_release = " "
    Path_debug = " "
    PATH = " "
  />
  <Other
    ATRIAHOME = " "
    MSDEVDIR = "%GMPS_TOP%/MSDEV60"
    include = " "
    lib = " "
  />
</SET_ENV>
```

This file has a tag "Path" which attributes defines the variable Path. The variable PATH gets the value of the attribute and dependent on the build mode the value of Path_release or Path_debug. If there is no attribute defined in PATH, gms_make works with the set environment variable.

In the tag "Other" you can define some more variables which are given to the system by gms_make. If you don't define any variables in "Other", gms_make works with the available system variables.

If the value of an attribute is one or more space characters (" "), the variable is empty during the runtime of gms_make. If there is nothing in-between the quote signs (""), no new variable is set and the existing environment variables do not change.

In the attributes you can use Windows- or Linux-Syntax depending to you platform. For example:

- Windows: %variable%
- Linux: \$variable

4.4.5 gms_targets.xml

The file gms_targets.xml includes all targets of the project. The file is modified by gms_register and gms_unregister. This file is not edited by the user.

Structure of gms_targets.xml

```
<gms_targets>
  <target
    name = "como.jar"
    type = "jar"
    dopath = "${GMPS_TOP}/como_do/dvp"
    srcpath = "${GMPS_TOP}/como/dvp"
    configfile = "${GMPS_TOP}/como/dvp/como.xml"
    target_release_dir = "${GMPS_TOP}/como_release/lib"
    target_do_dir = "${GMPS_TOP}/como_do/lib"
  />
</gms_targets>
```

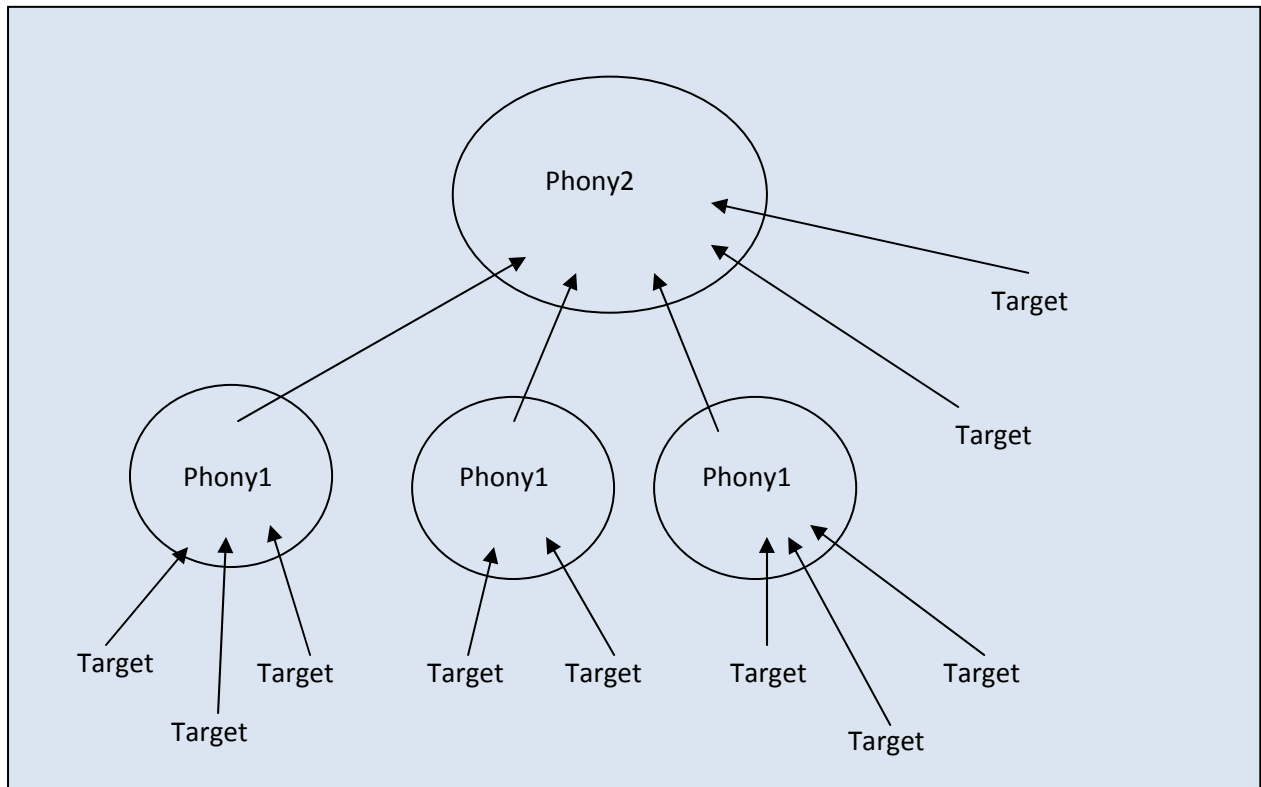
4.4.6 gms_magic.xml

The gmx_magic.xml file is responsible for the definition of the phonies.

A phony is an accumulation of targets. GMPS uses phonies to compile multiple targets at the same time. There are two kinds of phonies:

- Phony1 is a phony of the first level and it includes just targets.
- Phony2 is a phony of the second level and it includes phonies1 and targets.

An example of phony-structure:



Structure of gms_magic.xml:

```
<gms_magic>
  <phony2 name = "all">
    <target name = "tpsap"/>
    <phony1 name = "exe1"/>
    <phony1 name = "exe2"/>
  </phony2>
  <phony1 name = "exe1">
    <target name = "ex03a"/>
    <target name = "ex25b"/>
  </phony1>
  <phony1 name = "exe2">
    <target name = "ex11b"/>
  </phony1>
</gms_magic>
```

The order of the targets when you are generating a phony:

The targets inside of the phony are checked on direct dependencies and will be built in the right order.

In the gms_magic.xml file are only phonies allowed, no targets.

4.4.7 SetMacro.xml

SetMacro.xml file includes the macro definitions which are imported straight into the Makefile. The content of the “SetMacro” tag comes at the end of the Makefile.

Structure of SetMacro.xml:

```
<SetMacro>
Macro1,(e.g. “WATCOM_DIR = C:\path1”)
Macro2

</SetMacro>
```

“SetMacro” includes template definitions which are written for the Makefile.

4.4.8 gms_project_vobs.xml

The gms_project_vobs.xml file includes all VOBs of the project, excepting the GMPS-VOB. This file is used for labeling. If you call

```
gms_make -l Labelname
```

there will be created a new labeltype “Labelname” in this VOB and also in the GMPS-VOB.

4.4.9 template_section.xml

In template_section.xml are registered all XSLT templates.

Structure of template_section.xml:

```
<template name = “ “ file = “ “ />
```

Each new template has to be registered in this file.

4.4.10 target.options

target.option is a file in the source-directory, it has to be in the same directory as target.xml and can be modified by the user. It includes macro definitions which are coming after the build macro definitions in the Makefile structure. These definitions are overwriting the original definitions.

Structure of target.options:

```
<Target_options>
CPPFLAGS = -d redefinition
LD = /usr/local/Purify
</Target_options>
```

“target_options” includes a text with the synopsis of the Makefile. This file is only considered if you set the option `-mo` when calling `gms_make`.

5. Main programs of GMPS

The main programs of GMPS are:

Script	Purpose
<code>gms_make</code>	Create final Makefiles for all selected targets. Build selected targets or executes other commands specified as parameters.
<code>gms_register</code>	Create target specific sub trees in the do-directory and add them to source control. Create a new dataset for each target in <code>gms_targets.xml</code>
<code>gms_unregister</code>	Removes the dataset for a target from <code>gms_targets.xml</code> . Does not remove specific sub trees from the do-directory.

5.1 `gms_make`

`gms_make` is the central control program.

5.1.1 Synopsis Command line

General use in the command line:

```
gms_make Target [-debug | -release] [-mo] [-clean]
               [-platform win32 | -platform linux]
               [-make clearmake | - make make | - make omake]
               [-gf "make comandline options"] [-nm] [-l Labelname]
               [-lr Labelname] [-tag Name] [-tag_release Name]
               [-cld]
```

`gms_make`

Create final Makefile for all selected targets. Build selected targets or execute other commands (e.g. `make clean`) specified as parameters.

`Target` is a registered target in `gms_targets.xml` or a phony in `gms_magic.xml`.

`-debug`

Compile target in debug mode.

`-release`

Compile target in release mode.

`-mo`

Read and execute the `target.option` file.

`-clean`

Clean up all files created during generation of specified target.

`-platform win32`

Select the platform win32.

-platform linux

Select the platform linux.

-make clearmake | make | omake

Define the make tool. Default is clearmake.

-gf "make comandline options"

The make command line options will be imported straight into "make".

-nm

Generate only the Makefile.

-l Labelname

Just in ClearCase: label all associated sources from `target` with label `Labelname`; if the labeltype `Label` does not exist, `gms_make` creates the labeltype in all defined project VOBs.

-lr Labelname

Just in ClearCase: label all associated sources from `target` with label `Labelname`; if the labeltype `Label` does not exist, `gms_make` creates the labeltype in all defined project VOBs.

Check-in the `target` to the defined release area and label the checked data.

-tag Name

Just in Subversion: create a tag in the repository/`tags/Name` with the content of "`$(GMPS_TOP)/*directories`".

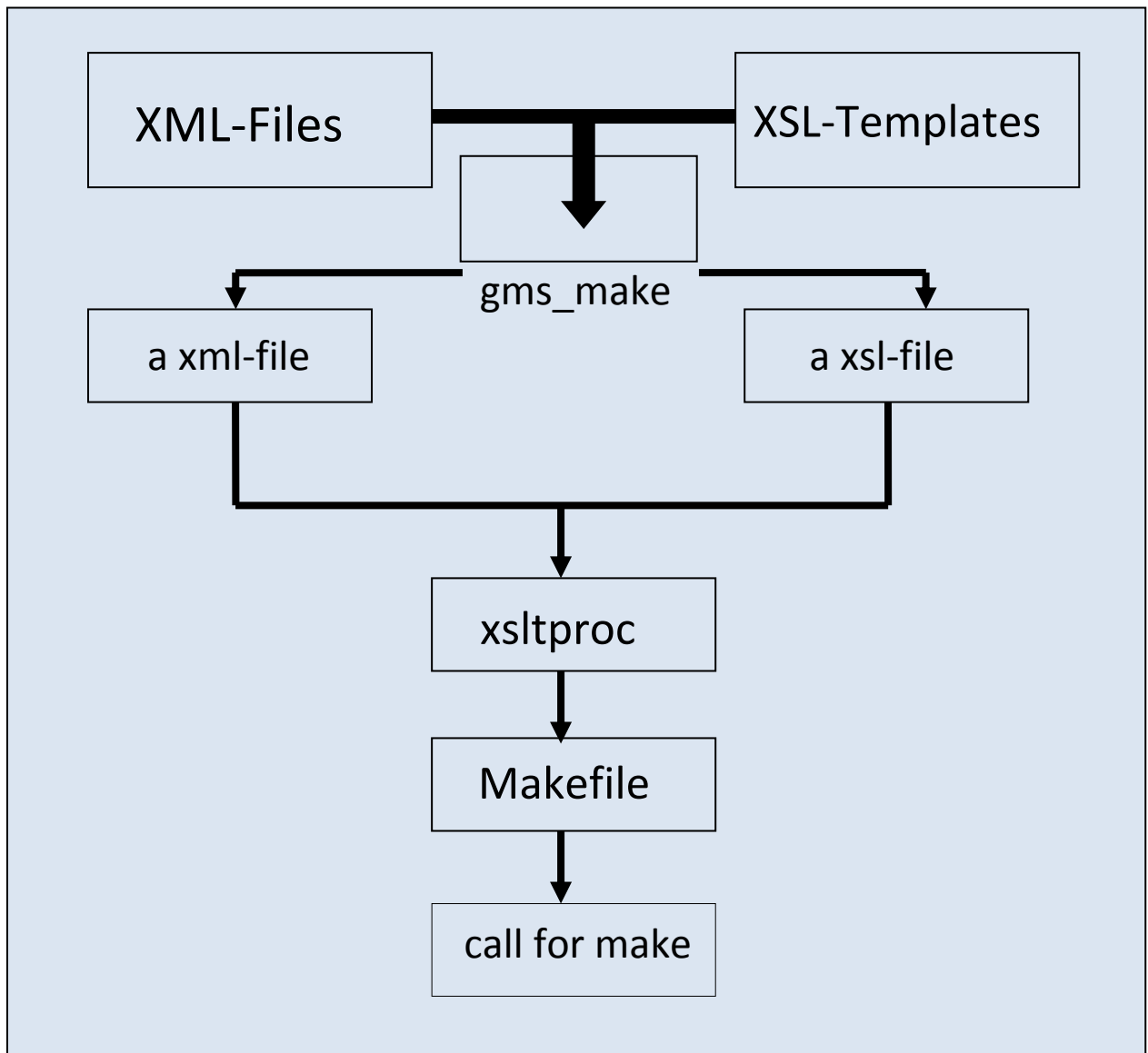
-tag_release Name

Just in Subversion: create a tag in the repository/`tags/Name` with the content of "`$(GMPS_TOP)/*directories`". Copy all generated `targets` in the `project_release` directory

-cld

Check load dependency. `gms_make` checks all load or link dependencies and builds the needed dependent libraries. This command is only valid for `targets`, not for `phonies`. If there is `<Depends Libs = " "/>` in `target.xml`, `gms_make` recursive checks each target in `Libs`, if it is a project target or not. If it is a project target it will be built according to its dependencies. If `<Depends Libs = " "/>` is not existing in `target.xml`, `gms_make` does not report any error and is only building the named `target`.

5.1.2 Workflow of gms_make



5.2 gms_register

The template `gms_register` updates the target registry file `gms_targets.xml` and modifies the do-directory structure if necessary. `gms_register` can only be started at the source location of your target. `gms_register` analyses the related `*.xml` file and depending on the `buildmacro` inside it registers different entries (like target type {`exe` | `dll` | etc.}, targets do/release area location).

5.2.1 Synopsis command line

General use in the command line:

```
gms_register [-f target.xml] [-info] [-nci] [-gen gen_path]
             [-dir directory] [-rec] [-h | -help] [-v | -verbose]
```

`gms_register`

Create target specific sub trees in the do-directory and add them to source control. Create a new dataset for each target in `gms_targets.xml`

`-f target.xml`

Use this option if there are multiple xml files in the source location.

`-info`

Output information about the entries for `gms_targets.xml` and the modifications that would be made on do-directory

`-nci`

Don't check-in the `gms_targets.xml` file after the new target is registered.

`-gen gen_path`

The path for the generated includes/sources, in case that the template generates includes or sources

`-dir directory`

Define directory to look for a GMPS target.xml file. You can use `-f` option.

`-rec`

Search recursive GMPS target.xml file in current directory or in directory defined in `-dir` option and register associated targets.

`-h | -help`

Print a help screen and exit.

`-v | -verbose`

Set to verbose mode.

5.2.3 Workflow of `gms_register`

The program `gms_register` creates the `gms_targets.xml` file if necessary and puts all target data in the file. It also creates the directory tree in the do-directory and in the release-directory. For Subversion `gms_register` creates `project_do-` and `project_release-` directories in the working copy. By default the program looks for GMPS target.xml (target configurations file) in the working directory.

Examples:

The module1.xml file includes the Buildmacro executable "aTarget"

The module1.xml file resides in : /sources/module1

Command line: `gms_register -f module1.xml`

Registers the target "aTarget" in the `gms_targets.xml` file and creates a mirror of the directory structure of the source tree into the do-directory /do/module1

The module2.xml file includes the Buildmacro: library "aLibName"

The module2.xml file resides in: /sources/module2/lib

Command line: `gms_register -f module2.xml`

Registers the target "aLibName" in the `gms_targets.xml` file and creates a mirror of the directory structure of the source tree into the do-directory /do/module2/lib

Job steps of `gms_register`:

1. `gms_register` read the command line options
2. If there is no `gms_targets.xml`, `gms_register` creates this file
3. `gms_register` read the `gms_config.xml` file and collects the build mode, the platforms, the Doroot and the SourceLocation.
4. `gms_register` checks if there is a xml-file with the structure `<GMPS_PROJECT> <Target/> </GMPS_PROJECT>` and reads the configurations of this file into an array `@target_daten`.
5. `gms_register` checks if the target is already registered.
6. If not, `gms_register` creates the do-directory. If the target is already registered, `gms_register` exits.
7. `gms_register` writes all information about the new target in `gms_targets.xml`.
8. The new target is now registered.

5.3 `gms_unregister`

The template `gms_unregister` updates the target registry file `gms_targets.xml`. All entries of the specified target: target-name, source location, location for the do's, etc. are removed from the central registry file `gms_targets.xml`. The modifications made to the do-directory structure are not removed.

5.3.1 Synopsis command line

General use in command line:

```
gms_unregister target [-h | -help] [-nci] [-all]
```

```
gms_unregister
```

Remove the dataset for a target from `gms_targets.xml`. Does not remove specific sub trees from the do-directory.

Target is a registered target in `gms_targets.xml` or a phony in `gms_magic.xml`.

-h | -help

Print a help screen and exit.

-nci

Just for ClearCase: don't check-in the modified gms_targets.xml file after unregistering is complete.

-all

Unregister all targets from the gms_targets.xml file.

6. Some more GMPS tools

6.1 gms_list

6.2 gms_list_dep

6.3 csproj2cfg

6.4 netparser

6.5 dvp2cfg

6.6 gms_mkproj

6.1 gms_list

General use in command line:

```
gms_list [-t target] [-l | -long] [-dep] [-h | -help]
```

gms_list

List all targets of a project from gms_targets.xml sorted by type.

-t target

List all information for one target.

-l | -long

List long information about all targets.

-dep

List only the dependencies of the targets. In combination with -t option list dependencies of the named target.

-h | -help

Print a help screen and exit.

Default

List registered targets of the project.

6.2 gms_list_dep

General use in command line:

```
gms_list_dep target [-nograph] [-vizfile target.xml file] [-cld]
                  [-nocld] [-d | -debug] [-T graphviz_out_file]
                  [-man] [-h | -help]
```

`gms_list_dep`

List defined phonies and dependencies

Target is a registered target in `gms_targets.xml` or a phony in `gms_magic.xml`.

`-nograph`

Suppress graphical display of dependencies. Only a vizfile is created.

`-vizfile target.xml`

Specify the graphviz output for the `target.xml` file. Defaulted to `<TMP folder>/gms_list_dep.<target>.graphviz.cfg`

`-cld`

Activate check load dependencies.

`-nocld`

Deactivate dependency check.

`-d | -debug`

Select the debug build mode.

`-T graphviz_out_file`

Set type of output file (default of `-T` option is `.jpg`). For all output formats read:

<http://www.graphviz.org/doc/info/output.html>

`-man`

Print the manual page and exit.

`-h | -help`

Print a help screen and exit.

6.3 csproj2cfg (does not work in Demo-Version)

`csproj2cfg` creates or updates the target specific xml-file for GMPS 5.0 from Microsoft Visual Studio C# project file settings. If no options are given, `csproj2cfg` looks for `csproj`-files in the current working directory. Option `-f` can be used to specify the MSVS project files in case of multiple `csproj`-files in the current working directory.

Just for ClearCase: the results of `csproj2cfg` are automatically checked-in into ClearCase.

Just for Subversion: the results of `csproj2cfg` are automatically added to the version control.

Use `-nci` option to suppress this, for ClearCase and Subversion.

Just for ClearCase: only in case of changes the target.xml file is updated (new version created). If the target.xml file did not exist, it will be added to the source control.

6.4 netparser (does not work jet)

6.5 dvp2cfg (does not work jet)

6.6 gms_mkproj (does not work jet)

7. GMPS and XSL-Templates

To edit the GMPS configuration files (tools.xml, target.xml, SetEnv.xml, SetMacro.xml, etc.) you need the XSL-templates which are included in the `$GMPSHOME/xsl-tmpl/` directory. The templates are defined in different files, so you can find and adjust them much easier.

The templates for the node of the target.xml file are defined in "match". The template "target" in the file target.xml edits the `<Target>`. This template sets up the macros `GMS_TARGET`, `VPATH`, `TargetSrcPath` and `TargetDoPath`. In the file there are two templates. One that edits tools.xml, the other one edits the flags for target.xml.

For the node of the sources and flags there are one or two xsl-files. E.g. `cpp.xml` and `cpp2obj.xml`. There are also templates that describe the Makefile, e.g. `cpp2obj.xml` and `rc2res.xml`. These templates are called by tools.xml. Further there are templates which edit the build nodes, e.g. "executable" or "dll". For the configuration of target.options and SetMacro.xml there are the templates `target_options.xml` and `SetMacro.xml`.

7.1 Define new templates

The user of GMPS can write new templates to support processes which are not included in the GMPS-package.

- 7.1.1 Modify target.xml
- 7.1.2 Modify tools.xml
- 7.1.3 Template for editing tools.xml
- 7.1.4 Write xsl-templates
- 7.1.5 Edit the file template_section.xml

7.1.1 Modify target.xml

Define the attribute "type" in the tag `<Target>` or use predefined types (exe, dll). Write the target name in the attribute "name".

```
<Target Type = "rpcgen" Name = "tpsap"/>
```

If there are sources which are important only for the build process and which don't need to be built before, than you have to define them in the build tag. E.g.:

```
<Buildtag
  XxSources = " "
/>
```

7.1.2 Modify tools.xml

All tools which can be used by a new target have to be defined in tools.xml. If one tool does not exist in tools.xml, you have to define a new tag for this tool. Description below, 7.1.3.

7.1.3 Template for editing tools.xml (documentation is not ready jet)

7.1.4 Write xsl-templates

```
<xsl:template match = "foo">
```

First you call the templates for editing the flags:

```
<xsl:call-template name = "tool_flags">
  <xsl:with-param name = "name" select = "'FOO'"/>
  <xsl:with-param name = "path" select = "'//FOO'"/>
</xsl:call-template>
```

This will generate FOOFLAGS, FOODEBUG, FOORELEASE.

```
<xsl:call-template name = "target_flags">
  <xsl:with-param name = "tool" select = "'FOO'"/>
</xsl:call-template>
```

This will generate FooFlags, FooRelease, FooDebug.

To read the sources:

```
XxSources=<xsl:value-of select = "@XxSources"/>
```

Files with the name of the target, which will be built by generation of the target and which will be deleted later with `-clean` option, can be written this way:

```
bar = <xsl:value-of select = "$target"/>.bar
```

`$target` represents the target name without extension. `$target` is a global xslt-parameter. In the Makefile it looks like this:

```
bar = TargetName.bar
```

Then you can see a Makefile-text, e.g.

```
$(GMS_TARGET) : $(XxSources)
  foo $(FOOFLAGS) $(FOODEBUG) $(FOORELEASE) $(FooFlags) $(FooDebug) $(FooRelease)
```

clean:

```
rm -rf $(bar)
rm -rf $(GMS_TARGET)
```

\$(GMS_TARGET) is the name of the target with extension, a predefined Makefile macro.

7.1.5 Edit the file template_section.xml

```
<template name = "foo" file = "foo.xml"/>
```

8. Index of xml-tags for the GMPS Target configuration file

- 8.1 C/C++ projects
- 8.2 JAVA projects
- 8.3 C# projects
- 8.4 Miscellaneous

8.1 C/C++ projects

Syntax of the tag	Description
<pre><Cpp GlobalSources = "" ReleaseSources = "" DebugSources = "" LinuxGlobalSources = "" LinuxReleaseSources = "" LinuxDebugSources = "" Win32GlobalSources = "" Win32DebugSources = "" Win32releaseSources = "" GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" sSuffix = "" oSuffix = "" Vpath = "" /></pre>	<p>sSuffix and oSuffix are necessary and have to be defined with a dot. E.g. sSuffix = ".cpp".</p> <p>If there are C and C++ sources the following syntax is required:</p> <pre><Cpp> <Cpp Attribute /><Cpp Attribute /> </Cpp></pre> <p>The flags need to be the same and don't have to be mutually exclusive, this is a premise for this syntax. Because the flags in each <Cpp> will be added to the same flag macro. So it's enough to define the flag attributes in one <Cpp>. This uses the C-tools and flags from tools.xml.</p>
<pre><Rc</pre>	Uses the RC-tool and flags from tools.xml

<pre> GlobalSources = "" ReleaseSources = "" DebugSources = "" LinuxGlobalSources = "" LinuxReleaseSources = "" LinuxDebugSources = "" Win32GlobalSources = "" Win32DebugSources = "" Win32releaseSources = "" GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" Vpath = "" /> </pre>	
<pre> <Midl IdlSource = "" NewHeader= "foo.h" NewInterface = " bar_i.c" NewTlb = "foobar.tlb" GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" /> </pre>	<p>In the attribute "IdlSource" is only written one IDL-file. It uses the MIDL-tool and the flags from tools.xml.</p> <p>The attributes "NewHeader", "NewInterface" and "NewTlb" are overwriting the default values.</p> <p>Basic names for IdlSource + End: .h, _i.c, .tlb</p> <p>You can't left these attributes empty if they are existing in the configuration file of the target. Each of this three attributes is optional. In the attributes you have to define the name and the ending of the file.</p>
<pre> <Dll DefSource = "" Libs = "" GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" </pre>	<p>This is a build tag.</p> <p>In DefSource you just have to define one DEF-file.</p> <p>Libs represent a list of libraries, separated by space character.</p> <p>All attributes are optional.</p>

<pre>Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" Libs = "" /></pre>	<p>Uses <LD> from tools.xml</p>
<pre><Dll_Lib GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" /></pre>	<p>A build tag.</p> <p>With it you can build a LIB for a dll.</p> <p>Disregards the LD-flags from tools.xml, just uses his own flags. It works with <LD> from tools.xml</p>
<pre><Executable GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" Libs = "" /></pre>	<p>A build tag.</p> <p>It builds executables and uses the LD-flags and LD-tool from tools.xml.</p>
<pre><Lib GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" /></pre>	<p>A build tag.</p> <p>Builds an import-lib. Uses AR-tool and flags from tools.xml.</p>
<pre><Library</pre>	<p>A build tag.</p>

<pre>GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" Libs = "" /></pre>	<p>It uses CC-tool and LD-flags.</p> <p>Libs is a list of libraries separated by space character.</p>
<pre><MakeDepend/></pre>	<p>Each file that includes a source file, direct or indirect, has a dependency according to MakeDepend.</p> <p>These dependencies are written in a Makefile. This Makefile can be used in the ClearCase Snapshot View or in Subversion.</p> <p>No attributes.</p> <p>Uses the flags from <CC> and <Cpp> from the target.xml file.</p>

8.2 JAVA projects

Syntax of the tag	Description
<pre><Javac ClassPathWin = "" ClassPathLinux = "" Dir = "" DirDebug = "" DirRelease = "" GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" /></pre>	<p>Tested with clearmake and omake.</p> <p>ClassPathWin describes the CLASSPATH for Windows. Directories are separated by semicolon.</p> <p>ClassPathLinux describes the CLASSPATH for Linux. Directories are separated by colon.</p> <p>Dir respectively DirDebug or DirRelease describes the source-directory with the absolute path. E.g. \$(GMPS_TOP)\path\to\sources.</p> <p>Uses JAVAC-tool and flags from tools.xml.</p>
<pre><Jar</pre>	<p>Tested with clearmake and omake.</p>

<pre> Pattern = "" PatternRelease = "" PatternDebug = "" GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" /> </pre>	<p>Syntax of pattern (PatternDebug, PatternRelease):</p> <p>Attribute: 'Directory "Pattern" [Directory "Pattern"]...'. E.g. <code>\$(GMPS_TOP)\como\divp "src\com*.properties"</code> <code>\$(GMPS_TOP)\como\divp "src\com*.image" \$(GMPS_TOP)\como\divp "src\com*.xml"</code></p> <p>Uses JAR-tool and flags from tools.xml.</p>
<pre> <JavaDoc Dir = "" DirDebug = "" DirRelease = "" GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" /> </pre>	<p>Tested with clearmake and omake.</p> <p>The xsl-template which edits this tag uses CLASSPATH from the tag <Javac>.</p> <p>Dir respectively DirDebug or DirRelease describes the source-directory with the absolute path.</p> <p>Uses JAVADOC-tool and flags from tools.xml.</p>

8.3 C# projects

Syntax of the tag	Description
<pre> <NetcsResource GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" > <NetcsResourcesEmbeddedRename </pre>	<p>In all arguments the name of the sources is written without ending.</p> <pre> <NetcsResourcesEmbeddedRename Source = " " Resource = " " /> </pre> <p>Source is the existing source name and Resource is the new name. In each source is only written one file name. So if you want to define more you have to recur the tag <NetcsResourcesEmbeddedRename> respectively <NetcsResourcesEmbeddedLizens> or <NetcsResourcesEmbedded>.</p>

<pre>Source = "" Resource = "" /> <NetcsResourcesEmbeddedLizens Source = "" /> <NetcsResourcesEmbedded Source = "" /> </NetcsResource></pre>	<p>It uses NETCSRES-flags and tool from tools.xml.</p>
<pre><NetcsDll GlobalFlags = "" ReleaseFlags = "" DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" /></pre>	

8.4 Miscellaneous

Syntax of the tag	Description
<pre><Target Name = "" Type = "" /></pre>	<p>The tag and the arguments are necessary.</p>
<pre><Depends Libs = "" /></pre>	<p>In the Libs attribute are defined the GMPS targets, separated by space character, which are dependent of a target. If this tag is defined in the target.xml file you can call the <code>-cld</code> option from <code>gms_make</code>.</p>
<pre><Zip ZipData = "" ZipDataDebug = "" ZipDataRelease = "" GlobalFlags = "" ReleaseFlags = ""</pre>	<p>A build tag.</p> <p>The target is the name of the zip-file.</p> <p>It generates a zip-file with files in "data" arguments.</p>

<pre> DebugFlags = "" LinuxGlobalFlags = "" LinuxReleaseFlags = "" LinuxDebugFlags = "" Win32GlobalFlags = "" Win32DebugFlags = "" Win32releaseFlags = "" /> </pre>	<p>It can use Wildcards like</p> <pre> ZipData = "\${GMPS_TOP}\como\cfg\nls*.txt" </pre>
--	---

9. Appendix

- 9.1 GMPS and xsltproc
- 9.2 GMPS and omake
- 9.3 GMPS and platforms
- 9.4 External tools

9.1 GMPS and xsltproc

The GMPS-package includes everything you need to run the xsltproc under Windows. You can find it in `$GMPSHOME\xsltproc\win32\libxslt\bin`.

You have to set up the path `$GMPSHOME\xsltproc\win32\libxslt\bin` in the PATH variable.

In many Linux distributions the xsltproc is already included, as it is used intensively from many programs which are running on Linux. A call of xsltproc in the command line should give a short output how to use the program. If there is no message appearing in you prompt, you have to install the xsltproc.

9.2 GMPS and omake

For Windows NT the omake macronames are case-insensitive by default. The `“.CASE_MACRO”`-directive sets up case-sensitive. Therefore you edit the file `$CChome/bin/make.ini.CASE_MACRO`. To switch-off case-sensitive you can use the `.NOCASE_MACRO`-directive. (See also “ClearCase omake Manual”, page 24, 66).

9.3 GMPS and platforms

The xsl-templates are supporting at the moment the platforms “win32” and “linux”. For more platforms you have to modify the templates of the flags and sources.

9.4 External tools

For the GMPS-package the following external tools have been used:

- xsltproc
- “str.split.function” xsl-function from www.exslt.org/str/index.html
- GNUmake
- Open Watcom
- JDK
- Graphviz