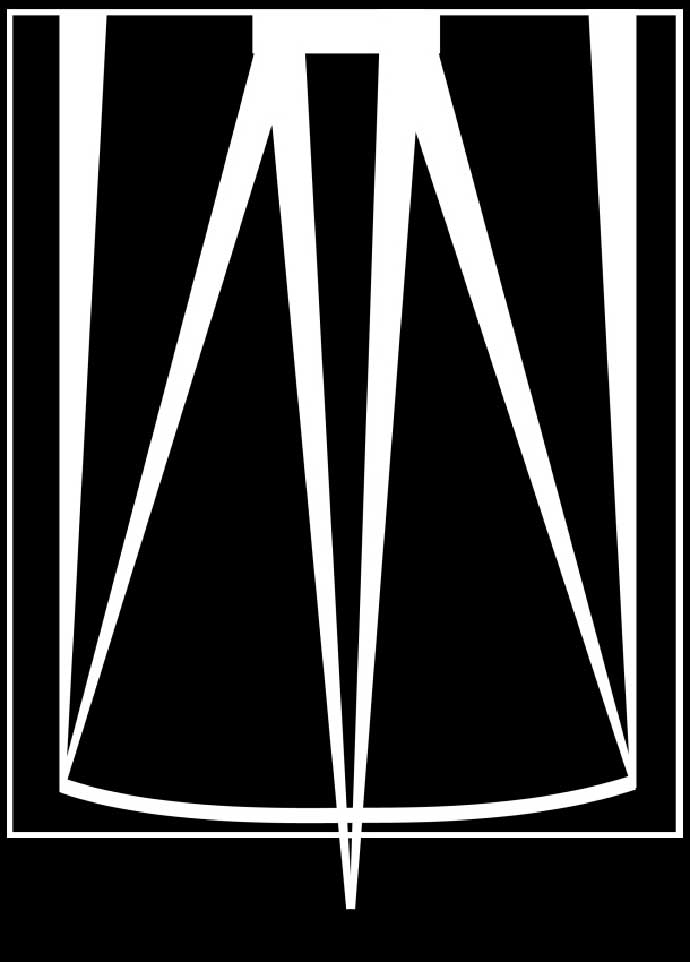
****

**Gemini Application**

**Development Environment**

**User Manual**

Issue 1.0

December 30, 2014

|  |  |  |
| --- | --- | --- |
| Author: | Philip Taylor, Observatory Sciences Ltd | pbt_signature |

Observatory Sciences Ltd

William James House

Cambridge. CB4 0WX.

United Kingdom.

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Authors | Date | Reason for Issue / Description of Changes |
| 1.0 | Philip Taylor | 30 December 2014 | Initial Release |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

[1 Introduction 5](#_Toc409109213)

[1.1 Purpose 5](#_Toc409109214)

[1.2 Acronyms and Abbreviations 5](#_Toc409109215)

[1.3 References 5](#_Toc409109216)

[1.4 Document Overview 5](#_Toc409109217)

[2 Gemini Real Time System Software Environment 6](#_Toc409109218)

[3 ADE Overview and Definitions 6](#_Toc409109219)

[3.1 ADE Directory Structure 7](#_Toc409109220)

[3.2 EPICS IOC Types 8](#_Toc409109221)

[3.3 IOC Naming 8](#_Toc409109222)

[3.4 EPICS Releases 9](#_Toc409109223)

[3.5 Categories of Built Software 9](#_Toc409109224)

[3.6 Vendor Software 10](#_Toc409109225)

[3.7 Software Dependencies 10](#_Toc409109226)

[3.8 Host and Target Architecture Builds 11](#_Toc409109227)

[3.9 Local Configuration Data 12](#_Toc409109228)

[4 Software Development process 13](#_Toc409109229)

[4.1 Normal Module Development 14](#_Toc409109230)

[4.2 Major Redevelopment 15](#_Toc409109231)

[4.3 Bug Fix to Released Code 15](#_Toc409109232)

[5 Development Examples 16](#_Toc409109233)

[5.1 Create new support module: install in work Area 16](#_Toc409109234)

[5.2 Create new IOC module: install in work Area 17](#_Toc409109235)

[5.3 Modify existing Modules installed in the work area 18](#_Toc409109236)

[5.4 Release support module to prod area 18](#_Toc409109237)

[5.5 Release IOC module to prod area 19](#_Toc409109238)

[5.6 Feature Branch : Major Redevelopment 19](#_Toc409109239)

[5.7 Bugfix Branch: Bug Fix to Released Code 20](#_Toc409109240)

[6 Booting IOCs 21](#_Toc409109241)

[6.1 configure-ioc boot configuration script 21](#_Toc409109242)

[6.2 Booting MVME6100 RTEMS IOC 22](#_Toc409109243)

[6.3 Booting MVME2700 RTEMS IOC 23](#_Toc409109244)

[7 ADE Directory Structure 24](#_Toc409109245)

[7.1 work Directory 25](#_Toc409109246)

[7.2 Production (“prod”) Directory 30](#_Toc409109247)

[8 ADE profile and Environment Variables 32](#_Toc409109248)

[9 Build & Release Scripts Reference 33](#_Toc409109249)

[9.1 gem-start-new-module.py 33](#_Toc409109250)

[9.2 gem-checkout-module.py 34](#_Toc409109251)

[9.3 gem-list-modules.py 35](#_Toc409109252)

[9.4 gem-changes-since-release.py 35](#_Toc409109253)

[9.5 gem-logs-since-release.py 36](#_Toc409109254)

[9.6 gem-release.py 37](#_Toc409109255)

[9.7 gem-start-feature-branch.py 38](#_Toc409109256)

[9.8 gem-start-bugfix-branch.py 39](#_Toc409109257)

[9.9 gem-sync-from-trunk.py 39](#_Toc409109258)

[9.10 gem-list-branches.py 40](#_Toc409109259)

[9.11 gem-vendor-import.py 40](#_Toc409109260)

[9.12 gem-vendor-update.py 41](#_Toc409109261)

[9.13 gem-get-vendor-current.py 42](#_Toc409109262)

# Introduction

## Purpose

This document provides a User Manual for the application development environment (ADE) which supports the Gemini Real-Time System software development process. For a detailed description, see reference [RD1].

Note that this document describes only the application development environment for the Gemini Real-Time systems. The development of Gemini instrument software is outside its scope.

## Acronyms and Abbreviations

This section gathers together some of the relevant acronyms and abbreviations used in this document

|  |  |
| --- | --- |
| ADE | Application Development Environment |
| EPICS | Experimental Physics and Industrial Control System |
| GUI | Graphical User Interface |
| IOC | EPICS Input/Output Controller |
| NFS | Network File System |
| OPI | Operators Interface (EPICS) |
| SVN | Subversion (version control system) |
| TBC | To Be Confirmed |

## References

*[RD1] Gemini Application Development Environment & Software Product Control/Release Procedures. Philip Taylor (Observatory Sciences Ltd). Version 1.0, dated 16 October 2014.*

## Document Overview

This User Manual describes most of the features of the Gemini Application Development Environment (ADE). Often there will be more information here than you need for your purpose. A quick summary of the contents follows, which should provide pointers for specific purposes.

1. **How do I use the Gemini ADE for software development?** If you are familiar with the basic concepts behind the ADE, Chapter 5 as well as Chapter 4 provides examples and background on the normal development procedures when using the ADE.
2. **What is the Gemini ADE?** Chapter 2 defines the ADE software and supported hardware; Chapter 3 provides information about the concepts and terminology used in the ADE.
3. **How do boot my EPICS IOC?** Chapter 6 describes how the ADE allows configuration of where an IOC boots from, as well as IOC boot parameters.
4. **Details of ADE directories, environment and scripts.** Reference details are provided in Chapter 7 (ADE directory structure), Chapter 8 (environment) and Chapter 9 (ADE Python scripts).

# Gemini Real Time System Software Environment

The Gemini Application Development Environment (ADE) comprises software and tools as follows:

1. **Development Workstation and EPICS Soft IOC platform**: 64 bit x86 instruction set (x86\_64 architecture) PC running CentOS Linux operating system, version 6.5.

CentOS Linux is a free distribution of Red Hat Enterprise Linux (RHEL), excluding Red Hat proprietary software and licensing restrictions.

1. **RTEMS 4.10 operating system** software, comprising the OS kernel and tools for cross-development of applications running under RTEMS, including a cross-compiler with support for the beatnik RTEMS BSP for the MVME6100 VME board and BSP support for the MVME2700 VME board (currently using the RTEMS standard mvme2307 BSP). The RTEMS “addon” and BSP extension packages must have been built and installed for the supported target boards.

The RTEMS software is installed in $GEM\_ROOT/targetOS/RTEMS.

1. **EPICS R3.14.12.4 Base** software, installed in the directory pointed to by the environment variable $EPICS. It should have been built to support the host (linux-x86\_64) architecture as well as the cross-compiled target architectures (RTEMS-beatnik and RTEMS-mvme2307).

The EPICS software is installed in $GEM\_ROOT/epics/R3.14.12.4.

1. Appropriate **EPICS extensions**, compatible with the Base EPICS installation, including but not confined to: TDCT v2.13.19 (or later), msi and EDM. These will be installed in the directory pointed to by the environment variable $EPICS\_EXTENSIONS.
2. Gemini ADE specific scripts and EPICS configuration files.
3. **Display Tools:** *GUI development is outside the scope of the ADE.*
4. **Configuration Management Tool:** Subversion (SVN). Version 1.6.11 is currently used.

# ADE Overview and Definitions

The Application Development Environment to support the Gemini Real-Time System software development process consists of

* A comprehensive directory structure which is maintained in a file system accessible to all developers and operational systems. This defines standard locations in which different types of software modules can be built and accessed using automated builds and scripts.
* The directory structure defines locations which reflect the following software characteristics:
  + Module type, based on its functionality. A software module is defined to be either a *Support Module* or an *IOC Application*.
  + Maturity. Built software is categorized as being either *Work* software or *Production* software depending on how stable and reliable it is considered, usually based on the level of testing that has been performed. Software is designated as Production software when it is released, following testing.
* A complete set of built software is kept up-to-date, with the software installed in standard locations, using an automated build server. Multiple versions of production software modules are maintained. This released (production area) software is protected with read-only access.
* A source code control system, based on Subversion (SVN), along with a set of Python scripts that work with Subversion to standardize the processes in the software development cycle.
* A Build system based on GNU Make (hereafter referred to as “Make” in this document). The standard EPICS Build conventions are adopted, with enhancements including new Make rules, additional templates, macros, configuration files and consistency checking features.
* IOC naming conventions and standard startup scripts which enable EPICS IOCs to be booted with any specified version of software, supported by associated scripts and files to enable easy changing of the IOC boot configuration.

Some additional features are:

1. Only source code, not built code, is stored in the Subversion repository.
2. Software is never copied directly from one area to another (i.e. from home to live testing or from the test area to production) – it is always moved to a new area via a commit and checkout from the Subversion repository. Commit and checkout are performed by Subversion scripts which have been written especially to support this process. This ensures that what is tested is reproducible from the repository.
3. Any software production release is tagged in the repository so it can be recovered directly from the tag.
4. No changes are required to source code to build and run it in any of the checked out locations.
5. In some cases of externally supplied software trees (e.g. EPICS and RTEMS), there is no process of moving from test to release version. A new version of EPICS might be tested and then declared to be the production version. New local application development is then done using the new EPICS tree.

## ADE Directory Structure

One of the most important features of the Gemini ADE is a well-defined directory structure for both source and executable files which reflects software dependencies, type and maturity. The ADE has a root directory, in this document this is **/gem\_sw** (shown as $ROOT below).

An ADE software module directory name has the following structure in the Gemini ADE:

$ROOT/<maturity>/<EPICSversion>/<module\_type>/<module\_name>

Subsequent sections provide a detailed description of the meaning of the module top-level name components. Below this level, software modules have a directory structure defined by the default EPICS build/release environment.

A few examples will provide an overview of these features:

1. /**gem\_sw/work/R3.14.12.4/support/slalib**

This software module is not yet released into production use and so it is in the work area of the ADE directory tree. It is the top-level directory of a support software module called slalib. It will be built using the files and libraries provided by EPICS release R3.14.12.4.

1. **/gem\_sw/prod/R3.14.12.4/support/timelib/1-8-6**

This software module has been released into production use as version 1-8-6 and so it is in the prod area of the ADE directory tree. It is the top-level directory of a support software module called timelib. It has been built using the files and libraries provided by EPICS release R3.14.12.4.

1. /**gem\_sw/work/R3.14.12.4/ioc/GEMTEST/MK**

This software module is not yet released into production use and so it is in the work area of the ADE directory tree. It is the top-level directory of an IOC software module with the IOC named as GEMTEST and the IOC location MK (Gemini North). It will be built using the files and libraries provided by EPICS release R3.14.12.4.

1. /**gem\_sw/prod/R3.14.12.4/ioc/GEMTEST/CP/1-1**

This software module has been released into production use as version 1-1 and so it is in the prod area of the ADE directory tree. It is the top-level directory of an IOC software module with the IOC named as GEMTEST and the IOC location CP (Gemini South). It will be built using the files and libraries provided by EPICS release R3.14.12.4.

## EPICS IOC Types

The ADE is designed to build and install software for all types of EPICS IOCS, including RTEMS (hardware) IOCs as well as Soft Linux IOCs.

A Soft IOC is an EPICS IOC running on a non-embedded Linux host. There may be more than one Soft IOC per Linux host. Normally it does not depend on any hardware on the host with the exception of standard communications interfaces such as Ethernet.

## IOC Naming

The operation of the ADE depends on a consistent naming scheme being adopted for all operational IOCs. The host name of the IOC must be set correctly to enable it to automatically locate and access the scripts and application software for use with the IOC.

Gemini IOCs have names of the form

<system>-<location>-IOC*-<number>*

Where

system : (string) The name of the software subsystem run on this IOC, usually the hardware that is being controlled. For example, the string AG would be used to denote the IOC controlling the Gemini A&G.

location: (string) The physical location of the IOC. For Gemini, this is either MK or CP, denoting Gemini North or South.

*number*: Optional (2 digit integer) number identifying the IOC in a set of multiple IOCs associated with this subsystem at this location. *At Gemini there will usually be only a single instance of an IOC controlling a particular piece of hardware at a given location, and so in most cases, this value will be omitted.*

A complete Gemini IOC name might therefore be: AG-MK-IOC, meaning the IOC running the A&G control software at Gemini North (MK).

The ADE includes automatic building of IOC startup scripts from a script “source file” which provides for substitutions of location and host specific strings. The startup script for an IOC is accessed via a single line redirection script at a fixed location. The executable file to run the IOC is accessed via a soft link with a name fixed for that IOC (see section 6).

## EPICS Releases

Ideally, only a single EPICS release would be in use with the Gemini ADE at one time. In reality, especially during periods of upgrading, multiple EPICS releases must be supported simultaneously.

The file structure of the ADE defines a dependency on the EPICS version used when building the software. A new branch of the ADE tree is created for each EPICS version. The ADE directory structure supports these dependencies by having the name of the EPICS release at a high-level in the hierarchy. All work or production release versions are therefore specific to the EPICS release with which they were built.

The EPICS release supported with the initial implementation of the Gemini ADE is the latest stable version: R3.14.12.4 (released on 16 December 2013).

## Categories of Built Software

Software releases are categorized in a way which is reflected in the standard ADE directory structure. They are categorized in two ways:

1. The type of software module. This can be either a *Support Module* or an *IOC Application*.
2. Categorized according to the degree of maturity of the released software, which depends on how well it has been tested and when and where it should be used. There are two categories of released software: *Work* and *Prod*(*uction*).

### Software Module Type

The following two types of software module are supported by the ADE

1. **Support Module**. This is a self-contained EPICS software module which is intended to be used by another application. Typically it will consist of the software that implements a specific control system (e.g. the TCS or ECS), provides EPICS device or driver support software or provides a software library. Support module software is located in a directory tree with its top-level called support, with a structure as described in section 7.1.1 below.

It has been decided that, at Gemini, files or software that implement the Graphical User Interface will not be part of the support module but will be implemented separately, outside the scope of this ADE.

1. **IOC Application**. This comprises the set of files which will be loaded and run on a specific IOC, making use of software from one or more Support Modules. The name of the application is based on the hostname of the IOC. So for example the IOC with hostname **AG-MK-IOC** runs the application called **AG-MK-IOCApp**.

IOC application software is located in a directory tree with its top-level called ioc. The IOC application source code area contains only files which are specific to an IOC. So, for example, software which is identical for IOCs running the same system (e.g. IOCs that run the TCS at Gemini North and South) would not be part of the IOC application, but static configuration data files which differ between the two sites *would* be included in the IOC application source code (see Section 3.9 below).

### Software Maturity

The Gemini ADE categorizes software modules according to their “maturity”, which is a measure of their apparent stability and reliability. Software will be considered mature when it has been extensively tested and is undergoing less rapid change due to fewer new features and bug-fixes.

1. **Unreleased** software is not handled as part of the development environment and does not have a location in the standard ADE directory structure. This is software in its initial development and test stage which is not yet available for use by other users. The development and test work will typically be performed in a developer’s home directory area. However, such software can be used when developing and testing an IOC application or support module in the ‘work’ area.

There are two maturity categories for released software supported by the ADE:

1. **Work** software. This software is in a state ready for initial use with the real hardware but is considered to be under test and not ready for final release. It would typically be used during commissioning or engineering periods but is not intended to be used during routine operations. It should be of a sufficient quality to allow it to be used in other IOCs applications other than the associated test application.

The work software will have been released from the user’s development area and installed in the appropriate directory tree (top-level work) in the standard directory structure. No version name is associated with software modules released in the work area, although at any time the software can be tagged in the SVN code repository.

1. **Prod(uction)** software**.** This software is in a well-tested state, ready for routine operational use. It will normally have been previously located in the work area. The software is released and installed, with an associated version name, in the appropriate directory tree (top-level prod) in the standard directory structure. The software is protected with read-only access. Multiple versions of production release software modules will be maintained simultaneously. Each version name corresponds to the tag with that name in the SVN code repository.

Production software should be only dependent on production modules i.e. there should be no dependencies on software in the ‘work’ area.

## Vendor Software

Vendor software is any software imported from outside Gemini: this may be EPICS modules or any other software. Vendor software can be managed within the ADE. Scripts are provided (see section 8) to handle vendor software within the SVN repository. An SVN branch is created i.e. "/vendor" where the source code is exactly as received from the 3rd party, before any local modifications might be made.

## Software Dependencies

An IOC application (or another support module) will usually depend on specific versions of one or more software support modules. It will always depend on the specific version of EPICS. The specific module version could be either:

* A released version stored within **prod** e.g. streamDevice version 2.2 in /gem\_sw/prod/R3.14.12.4/support/streamDevice/2-2.
* A version in **work** e.g.the currently version of AG being worked on, in /gem\_sw/work/R3.14.12.4/support/AG
* A version within a private directory e.g. an initial version of a new module, under test in the developer’s home directory:

/home/ptaylor/devel/tcs

Wherever possible, a released version stored within **prod** should only be dependent on other production modules. A file called **RELEASE** in the application’s **configure** directoryis used to define the modules being used and their locations.

The file contains a list of macros which define file paths pointing to the top-level directory of the required support modules. If we assume that the module being built references support modules called modA, modB and modC, the build system passes all these paths to the different tools used in building an IOC e.g.

* The tool which creates single executable file by linking to libraries from modA, modB and modC.
* The tool which creates a single, master dbd file by combining the “dbd” files from modA, modB and modC.
* The Makefile creates “db” file(s) for the IOC by copying files from the built “db” directories of modA, modB and modC and placing the result in the IOC’s “db” directory.

The potential exists for conflicting dependencies, for example where a module has a dependency on multiple modules, which have dependencies on different versions of a single module. In principle, only a single, specified version of a dependent module should be used everywhere when a module is built. Such conflicts are automatically detected and flagged at build time.

### Example

Let us assume we are building a development version of an IOC application (in **work**) and that it depends on support modules “modA”, “modB” and “modC”. modA is being developed within a developer’s home directory, modB is a version in **work** and modC is a released version (x-y) in **prod**.

In this case **configure/RELEASE** might contain the lines

SUPPORT=/gem\_sw/prod/R3.14.12.4/support

WORK=/gem\_sw/work/R3.14.12.4/support

HOME=/home/ptaylor/devel/AG

MODA=$(HOME)/modA

MODB=$(WORK)/modB

MODC=$(SUPPORT)/modC/x-y

## Host and Target Architecture Builds

The only host architecture supported by the Gemini ADE is Linux 64-bit CentOS (EPICS architecture name linux-x86\_64). The host machines, in particular the build server, must be compatible with the ADE tools, such as the scripts and compilers provided. Multiple target architectures are supported by the directory structure: Linux 64-bit and RTEMS using Board Support packages (BSPs) for MVME2700 and MVME6100 PowerPC boards.

When EPICS is built, the file $EPICS\_BASE/configure/os/CONFIG\_SITE.<hostarch>.Common defines the list of target (cross-compilation) target architectures using the variable CROSS\_COMPILER\_TARGET\_ARCHS. Both target architectures should be defined here and EPICS will be built for the host and both targets.

By default, any application build will build for the cross-compilation architectures defined in EPICS Base. To restrict your application build to one or the other (or neither), edit the local configure/CONFIG\_SITE file in your application module to redefine the value of the CROSS\_COMPILER\_TARGET\_ARCHS variable.

For example, to build only for RTEMS MVME2700, the line in configure/CONFIG\_SITE would be:

CROSS\_COMPILER\_TARGET\_ARCHS = RTEMS-mvme2307

To build only for the host (no cross compilation targets), the variable should be set to blank, so the line in configure/CONFIG\_SITE would be:

CROSS\_COMPILER\_TARGET\_ARCHS =

## Local Configuration Data

Many of the real-time Gemini control systems operate at both North and South sites, controlling almost identical hardware at each location. Inevitably, there will be differences between the two sites, an obvious example being the geographical location (latitude, longitude and altitude). There will also be different operational parameters as well as hardware differences between the two sites.

Although, in principle, the operational software for a given control system should be identical at Gemini North and South, in practice different versions of the same software modules have evolved over the years. The intention is that software source code variations between North and South systems will be eliminated as part of future upgrade work and any location specific system differences will instead be implemented using configuration data, as described below.

### Static and Dynamic Configuration

Configuration data will be used by software running on the Gemini IOCs at Gemini North and South. Typically, the data will be read from files by application software on startup. We define two types of configuration data, as follows:

1. **Static Configuration Data**. This is data that changes rarely, perhaps only after a significant change, or new feature added, to the system. Some location specific configuration data will never change – for example, the site latitude and longitude. Files containing this type of data are managed within the ADE, stored under SVN as part of the IOC application and, when in production, associated with a specific release. Changes in static data will require a new release of the IOC application.
2. **Dynamic Configuration Data**. This is data that changes more frequently, perhaps with each telescope run or even on a nightly basis. Changes in dynamic data *will* *not* require a new software release. Dynamic data will not be stored as part of a support module or IOC application and so is not handled within the ADE. The data files will be stored in a separate area under SVN.

# Software Development process

The development process to be adopted for a software module which is used by the Gemini Real Time systems is summarized in below. The software module may be either a support module or IOC application. Almost all interactions with the SVN repository are performed using the ADE Python scripts (named gem-\*.py), not with direct SVN commands.



Figure Gemini ADE Software Development Process.

When developing software using the SVN repository, there are typically three different scenarios we encounter. We will consider each one of these in turn.

In the following descriptions, ADE Python script names are displayed in bold italic e.g. ***gem-checkout-module.py***. These scripts are described in more detail in Chapter 8.

## Normal Module Development

This is the steady development of a support module or IOC application along a linear development/release schedule.

### Initial Development

* For the case where we are modifying an existing support module or IOC application (bug-fixes or adding new features), development begins by running ***gem-checkout-module.py***.
* For the case where we are creating a completely new support module or IOC application, development begins by running ***gem-start-new-module.py***.
* If the code for a new module has already been supplied by an external vendor it should be imported using .

The first stage of the development process will typically take place in a sub-directory of the developer’s home directory. Testing of an IOC application will involve using a development IOC and changing the boot parameters on that IOC to point at the built startup script located in the “bin/<target architecture>” directory underneath the top-level of their application. The user will need to edit “configure/RELEASE” to define the software dependencies, following the guidelines in section 3.7. Commits to Subversion can be done at any time to provide traceability.

### Development in ‘work’

When the new version on trunk is ready it can be checked out into **work** for testing on an operational IOC. ***gem-checkout-module.py*** is run to check-out the support module or IOC application from the trunk into the **work** tree. For an IOC application, the script: *configure-ioc* should be run to modify the IOC boot script location (see section 6.2) and set this particular IOC to boot from the **work** version. The user may need to edit “configure/RELEASE” to define the software dependencies, following the guidelines in section 3.7. Commits to Subversion can be done at any time to provide traceability.

### Decision Time

At the end of development in **work**, the application will either be ready or not ready to be released to the **prod** area. In either case, the *configure-ioc* script must be run again so that the production IOC boots from the **prod** tree i.e. it must not be left pointing at the **work** tree once the testing is finished

### Release to Production

When the system is considered to be ready for release, ***gem-release.py*** is run to generate a release tag and export the application to the **prod** area. If possible, ***gem-release.py*** will also do a test build first, before scheduling the module to be checked out and built in **prod** by the build server. The next time the build cron job runs on the build server, it will notice a scheduled build, check the module out and build it.

Which Linux version to use is determined by the EPICS release – the build process associates a particular EPICS release with a specific Linux version. No manual editing of the “configure/RELEASE” file is required at this stage (the release process will edit it if you are releasing for a different EPICS version than specified in the Subversion version of the file). When the production IOC is directed to use the newly released module using a new boot link and rebooted, it will pick up the new release.

## Major Redevelopment

This occurs when a significant change is required to a support module or IOC application and we expect this to take a long time to implement. During this period of time, we will still need to be able to modify the existing support module or IOC application as a result of user demands, including bug fixes. A “feature branch” is therefore created which will be used to develop and test the application including the new feature.

This process is basically the same as Normal Module Development described in section 4.1, for the first phase “Initial Development”.

However in this case development begins by running ***gem-start-feature-branch.py*** *rather* than***gem-start-new-module.py*** *or* ***gem-checkout-module.py***. Also ***gem-sync-from-trunk.py*** should be run periodically to keep the code up to date with what is in trunk. When complete the feature branch should be merged back into trunk using the “*svn merge*” command.

## Bug Fix to Released Code

This occurs when a support module or IOC application has been released to production (the **prod** area), some time has elapsed and bugs have been found. We have to fix the bugs for this particular release, despite the fact that further development and later releases of this support module or IOC application have been made since the release.

This situation is not intended to happen in normal circumstances and it should be avoided where possible.

This process is basically the same as Normal Module Development described in section 4.1, for the first phase “Initial Development”.

However in this case development begins by running ***gem-start-bugfix-branch.py*** *rather* than***gem-start-new-module.py*** *or* ***gem-checkout-module.py***. Unlike the Major Development process the script ***gem-sync-from-trunk.py*** is not to be used. Also when running ***gem-release.py***  the ***–b***  option must be used.

The developer must check that this bug is fixed in the trunk as well as in this branch. The branch should be deleted after the bug has been fixed in the trunk.

# Development Examples

The following examples illustrate the development of applications using the Gemini Application Development Environment. The development cases are based on those presented above in Section 4 (Software Development process).

It is assumed that the standard Gemini ADE development platform (PC running the CentOS 6.5 operating system) is used and that the application is to be cross-compiled and run on either an MVME6100 or MVME2700 VME board, running the RTEMS real-time operating system.

## Create new support module: install in work Area

In this case we are creating a new support module called testsupp and building an initial version in the work area.

An example support module called adeTest is supplied with the Gemini ADE (released in the directory TestApps/support) and software from this module is used in this example.

1. Create a new directory in your development area. From this new directory, issue the following command to create a new support module, using a standard ADE template

$ gem-start-new-module.py testsupp

This will create a new, minimal, support module using the template files. It will have little or no functionality. A functional EPICS module, including a database with supporting software, should be created by developing the required files – this is described in the following section.

1. Populate and modify files, as necessary, in the directories of the new support module application area (testsuppApp). The relevant files & directories to be copied over (or created) are
   1. The file configure/RELEASE that defines software dependencies, can be copied from directory adeTest/configure. The example support module adeTest has dependencies on the slalib, timelib, genSub and sncseq (EPICS sequencer) modules. Ensure that the specified versions of the modules, as defined in this file, do exist in the stated locations.
   2. The testsuppApp/src directory. The C, SNC source code (.st ) files and the associated Makefile can be copied from directory adeTest/adeTestApp/src.
   3. Edit the file src/Makefile as follows

* Change the support library (target LIBRARY\_IOC) name so that it has the same name as the application (testsupp).
* Change the .dbd file (target DBD) name so that it has the same name as the application (testsupp.dbd).
* Edit all the target name macro prefixes to be testsupp\_.
  1. The testsuppApp/Db directory, where database schematic (.sch) files and symbol (.sym) files are located, for use with TDCT. These files and the associated Makefile can be copied from directory adeTest/adeTestApp/Db.

1. Build the testsupp support module software by issuing a make command from its top-level directory.
2. Assuming the software builds OK (without error), all new and modified files should be committed in the testsupp module in the Subversion repository. Ensure that only source files, not executables or binaries, are added to the repository.
3. Using ***gem-checkout-module.py***, the module should then be checked out into the **work** area for testing on an operational IOC. For this module, the location would be /gem\_sw/work/R3.14.12.4/support/testsupp

The script ***gem-checkout-module.py*** checks out the files into the current working directory, so the commands would be:

$ cd /gem\_sw/work/R3.14.12.4/support

$ gem-checkout-module.py testsupp

1. Build the support module software by issuing a make command from the top-level directory of the checked-out application in the **work** area i.e.

$ cd /gem\_sw/work/R3.14.12.4/support/testsupp

$ make

## Create new IOC module: install in work Area

In this case we are creating a new IOC application module called GEMTEST-MK-IOC and making an initial installation in the work area.

An example IOC application module called PBT is supplied with the Gemini ADE (released in the directory TestApps/ioc) and software from this module is used in this example.

1. Create a new directory in your development area. From this new directory, issue the following command to create a new IOC application module, using a standard ADE template

$ gem-start-new-module.py –i GEMTEST/MK

This will create a new, minimal, IOC application module using the template files. It will have little or no functionality.

1. Populate and modify files in the local directories in the iocBoot directory and the App area (GEMTEST-MK-IOCApp) files, as necessary. The relevant files & directories to be copied over (or created) are
2. Edit the file configure/RELEASE that defines software dependencies. All modules used in the IOC should be specified here. Ensure that the software modules have their locations specified correctly.
3. The App/src directory. This will contain a standard C++ source file called, in this case, GEMTEST-MK-IOCMain.cpp. This file should not be modified.

The Makefile should be modified to define the support module files required by this IOC. In particular, the target macros GEMTEST-MK-IOC\_DBD and GEMTEST-MK-IOC\_LIBS should be redefined, adding lines which name the dbd and library files from the relevant support modules.

1. The App/Db directory. The Makefile should be modified to define the built database (.db) files that will be loaded on this IOC.
2. The App/config directory. This directory contains configuration data files which are specific to this IOC. A pvload specification (.pv) file should be created in this directory and the Makefile edited to use this file when building in this directory.
3. The iocBoot directory, with subdirectory iocGEMTEST-MK-IOC will contain the source startup script for this IOC, which in this case will be called stGEMTEST-MK-IOC.src. Edit the script and Makefile to ensure that the correct IOC name is used.
4. Build the IOC application software by issuing a make command from its top-level directory.
5. If you wish to test the IOC booting directly from the IOC application software module built in your own development area, then the full boot script path will need to be specified with the configure-ioc script.

For example, to boot the IOC called GEMTEST-MK-IOC from the private directory area /home/ptaylor/gemdev, then the configure-ioc script would be called with the –b option specifying the boot script path, as follows

$ configure-ioc –b /home/ptaylor/gemdev/GEMTEST/MK/bin/<arch>/stGEMTEST-MK-IOC.boot \

GEMTEST-MK-IOC

where <arch> is the RTEMS architecture of the IOC, either RTEMS-beatnik or RTEMS-mvme2307.

1. Assuming the software builds OK, any altered files should be committed to the Subversion repository.
2. The IOC application module should be checked out from the Subversion repository into the appropriate directory in the **work** area, for testing on an operational IOC. In accordance with the ADE directory naming conventions, in this case the directory will be /gem\_sw/work/R3.14.12.4/ioc/GEMTEST/MK.

The script called ***gem-checkout-module.py*** is used to check-out the IOC module from the trunk into this directory, using the following command

$ gem-checkout-module.py –i GEMTEST/MK

1. Build the IOC application software by issuing a make command from the top-level directory of the checked-out application in the **work** area.
2. Having built the IOC application module in the **work** area, the configure-ioc script should be used to tell the IOC to boot from the **work** directory. The command to define the boot location is:

$ configure-ioc –t work GEMTEST-MK-IOC

Further information about the boot conventions used by the ADE (the redirector system) and the configure-ioc script can be found in section 6.1.

1. Reboot the IOC called GEMTEST-MK-IOC and ensure that it operates as expected. The IOC should boot without error and issuing the IOC Shell command dbl should show the expected database records.

## Modify existing Modules installed in the work area

In this case we are modifying an existing support module or IOC application (bug-fixes or new features), development begins by running ***gem-checkout-module.py***.

The project is checked out of the SVN repository into a sub-folder of the current working directory.

To check out a support module called testsupp:

$ gem-checkout-module.py testsupp

To check out an IOC application module called GEMTEST-MK-IOC:

$ gem-checkout-module.py –i GEMTEST/MK

Software development then proceeds in the same way as for new modules, as described above in sections 5.1 and 5.2 with the software checked out and built in the same **work** area and the IOC configured to boot from there.

## Release support module to prod area

When a support module in the **work** area is considered properly tested and reliable enough for production use, it should be released as a named version in the **prod** area.

1. The gem-release.py script is used to release a support module to the **prod** tree. The released software is tagged in the release area of the repository and the build server then checks out and builds the specified version in the appropriate location in the **prod** area.

To release a version, which we will call 1-0, of the testsupp support module, issue the command:

$ gem-release.py testsupp 1-0

1. The build server will build the support module version 1-0 in the read-only directory

/gem\_sw/prod/R3.14.12.4/support/testsupp/1-0

1. The gem-release.py script starts by running a test build on the local machine: if this fails, the build server will not attempt to perform the build. If the build server runs the build but it fails with errors, then an email will be generated containing the error messages from the build procedure. Error messages will be seen in the files build\*.log and build\*.err in the software build directory.

## Release IOC module to prod area

When an IOC application module in the **work** area is considered properly tested and reliable enough for production use, it should be released as a named version in the **prod** area.

1. The gem-release.py script is used to release an IOC application module to the **prod** tree. The released software is tagged in the release area of the repository and the build server then checks out and builds the specified version in the appropriate location in the **prod** area.

To release a version, which we will call 1-0, of the GEMTEST-MK-IOC IOC module, issue the command:

$ gem-release.py –i GEMTEST/MK 1-0

1. The build server will build the GEMTEST-MK-IOC IOC module version 1-0 in the read-only directory

/gem\_sw/prod/R3.14.12.4/ioc/GEMTEST/MK/1-0

1. It is intended that IOCs installed in the prod area should normally have dependencies only on support modules in the prod area. Having successfully built and released all the support modules used by the IOC GEMTEST-MK-IOC, the configure/RELEASE file should be edited accordingly.
2. Having successfully released the IOC application module to the **prod** area, the configure-ioc script should be run to define the IOC boot script location (see section ), setting the IOC to boot from the **prod** area version 1-0. The IOC must have its hostname set to GEMTEST-MK-IOC. The procedure is similar to that described above in section 5.2, but with the released version of the IOC specified. In this case, the command to define the boot location is:

$ configure-ioc –v 1-0 GEMTEST-MK-IOC

## Feature Branch : Major Redevelopment

When a significant change is required to a support module or IOC application and we want to be able to continue modifying the existing support module or IOC application, a “feature branch” can be created which will be used to develop and test the application including the new feature.

1. The development of the branch is begun by running ***gem-start-feature-branch.py***. The project is checked out of the SVN repository into a sub-folder of the current working directory. To create a “feature branch” (called Branch1) of the support module testsupp, issue the command

$ gem-start-feature-branch.py testsupp Branch1

1. To create a branch (called Branch1) of the IOC application GEMTEST-MK-IOC, issue the command

$ gem-start-feature-branch.py –i GEMTEST/MK Branch1

1. Software development then proceeds in a similar way as for new modules, as described above in sections 5.1 and 5.2. However, when the script ***gem-release.py*** is used to release the module, the ***–b***  option must be used. The command to release the module as version named 1-0B1from the Subversion feature branch named Branch1, rather than from the trunk would be:

$ gem-release.py –b Branch1 testsupp 1-0B1

1. The script ***gem-sync-from-trunk.py*** should be run periodically to keep the code up to date with what is in trunk. When work is complete, the feature branch should be merged back into trunk using the svn merge command.

## Bugfix Branch: Bug Fix to Released Code

This is the case where a support module or IOC application was previously released to production (the **prod** area), and bugs have been found. Further development and later releases of the support module or IOC application have been made since the release, but we need to fix the bugs in the original release.

The script gem-start-bugfix-branch.py is used to create a “bugfix branch” from a specified, old version of the module.

Example

1. To create a bugfix branch of a previous release (version 1-0) of the IOC module GEMTEST-MK-IOC, issue the command:

$ gem-start-bugfix-branch.py -i GEMTEST/MK 1-0 1-0Branch

This will create the directory

$SVN\_ROOT/gem/branches/ioc/GEMTEST/MK/1-0Branch from

$SVN\_ROOT/gem/branches/ioc/GEMTEST/MK/1-0

and then checks out the branch into “./1-0Branch”.

1. Software development then proceeds in a similar way as for new modules, as described above in sections 5.1 and 5.2.
2. When the script ***gem-release.py*** is used to release the module, the ***–b***  option must be used to specify the bugfix branch name, in the same way as for a feature branch, as described above in section 5.6. The command to release the module as version named 1-0B1from the Subversion bugfix branch named 1-0Branch, rather than from the trunk would be:

$ gem-release.py –b 1-0Branch testsupp 1-0B1

# Booting IOCs

RTEMS IOCs will boot by downloading the executable and EPICS startup scripts using tftp (Trivial File Transfer Protocol) over the network from the host Linux PC. The tftp server on the Linux host will need to be configured to allow access to the IOC executable file and EPICS startup script in the Gemini ADE tree structure.

A summary of the settings for booting MVME2770 and MVME6100 cards over the network is shown below. For further details, see the firmware documentation provided by the original card manufacturer (Motorola) and hardware configuration details documented by Gemini.

## Boot Redirector

The Gemini ADE has a convention (the “redirector”) using a soft file link with a fixed name to point to the actual boot script file currently being used. The link has the same (fixed) name as the IOC being booted but points to a (variable) boot script file name which depends on the version of the IOC software being used. So, if the IOC application module has been built in the directory /gem\_sw/work/R3.14.12.4/ioc/GEMTEST/MK

then the name of the boot script will be

/gem\_sw/work/R3.14.12.4/ioc/GEMTEST/MK/bin/<arch>/GEMTEST-MK-IOC.boot.

where <arch> is the RTEMS architecture of the IOC, either RTEMS-beatnik or RTEMS-mvme2307.

Using the configure-ioc script, the boot link is created in the (restricted access) directory /gem\_sw/prod/redirector. As long as the software has been built in the standard work or prod areas, using this script hides the details of the boot script name & location from the user.

For consistency, the IOC should have its hostname defined as GEMTEST-MK-IOC.

## configure-ioc boot configuration script

You might sometimes want to boot an IOC application in work and later from a specific released version in prod. This is configured using a soft link target and startup script, which are defined using the configure-ioc command. Various options are available for this command and many options will either be defaulted or prompted for. Examples:

1. Create (or modify) the IOC configuration for IOC AG-MK-IOC. Configure it to boot from version 2-8 in the production area (this is the default area).

configure-ioc –v 2-8 AG-MK-IOC

1. Configure IOC AG-MK-IOC to boot from its work area.

configure-ioc –t work AG-MK-IOC

1. Show the current boot configuration for the specified IOC:

configure-ioc -l AG-MK-IOC

1. List all configured IOCs:

configure-ioc -L

The configure-ioc script always checks for the existence of the link and startup script file and will normally prompt if existing files are to be overwritten. The –f flag forces overwrite without prompting.

The IOC host name is checked to see whether it is a recognized host name and asks for confirmation if it is not known.

The full list of options for the configure-ioc command is shown in the following table.

|  |  |
| --- | --- |
| configure-ioc -h | Display help on command options. |
| configure-ioc -L | List all existing IOC boot soft links. |
| configure-ioc -l ioc | List the entry for specified IOC. |
| configure-ioc [options] ioc | Create or replace the boot link entry for the IOC named ioc.  where ioc = IOC host name and options are as shown below |
| -t area | Set the software area (maturity) to be work or prod.  If no area is specified, you will be prompted (default prod).  If prod (the default) is used, a version should be defined using the –v option. |
| -v version | Define version number. When this option is used, the prod area is assumed. |
| -a arch | Set the IOC architecture for use with -i (e.g.RTEMS-beatnik).  If arch is not specified, you will be prompted. |
| -e evers | Specify the version of EPICS to be used.  If this is not specified, the value of the environment variable $GEM\_EPICS\_RELEASE will be used. |
| -s script | Provide the full boot script name (full path). |
| -b boot | Provide the full boot file name (full path). |
| -f | "force" flag: if this flag is used, then the links will be created even when the boot or script files do not exist. |

## Booting MVME6100 RTEMS IOC

We assume that RTEMS IOC’s running on MVME6100 cards will have the Motorola MOTload firmware installed and this will be used for booting RTEMS. Note that the card may need re-configuring to use the correct Boot Flash Bank.

The appropriate MOTload global environment variables should be setup to boot using tftp with appropriate boot parameters defined. For example, if the Linux host has IP address 192.168.0.59 and the RTEMS IOC (an MVME6100) with hostname AG-MK-IOC has IP address 192.168.0.65:

MVME6100> gevShow

mot-script-boot

tftpGet

go -a006B7000

mot-/dev/enet0-cipa=192.168.0.65

mot-/dev/enet0-sipa=192.168.0.59

mot-/dev/enet0-gipa=192.168.0.1

mot-/dev/enet0-snma=255.255.255.0

rtems-client-name=AG-MK-IOC

epics-script=/gem\_sw/redirector/AG-MK-IOC.cmd

epics-nfsmount=192.168.0.59:/gem\_sw /gem\_sw

mot-/dev/enet0-file=/gem\_sw/redirector/AG-MK-IOC

Refer to the Motorola MOTload firmware manual for further details of the standard global environment boot parameters. The environment variables named ‘epics-\*’ are used by EPICS at boot time to define the EPICS startup script and the NFS mount command. The Linux host must be configured so that /gem\_sw directory can be mounted by the RTEMS IOC using NFS.

## Booting MVME2700 RTEMS IOC

We assume that RTEMS IOC’s running on MVME2700 cards will have the Motorola PPC1-Bug firmware installed and this will be used for booting RTEMS. Note that the card may need re-configuring to use the correct Boot Flash Bank.

The PPC1-Bug environment (ENV command) variables need to be configured correctly and the niot command used to boot using tftp with appropriate boot parameters defined. For example, if the Linux host has IP address 192.168.0.59 and the RTEMS IOC (an MVME2700) with hostname AG-MK-IOC has IP address 192.168.0.66, use the niot command from PPC1-Bug to set the network boot parameters:

PPC1-Bug>niot

Controller LUN =00

Device LUN =00

Node Control Memory Address =03F9E000

Client IP Address =192.168.0.66

Server IP Address =192.168.0.59

Subnet IP Address Mask =255.255.255.0

Broadcast IP Address =192.168.0.255

Gateway IP Address =192.168.0.1

Boot File Name ("NULL" for None) =/gem\_sw/prod/redirector/AG-MK-IOC

Argument File Name ("NULL" for None) =172.16.5.126:/gem\_sw:prod/redirector/AG-MK-IOC.cmd

Boot File Load Address =001F0000

Boot File Execution Address =001F0000

Boot File Execution Delay =00000000

Boot File Length =00000000

Boot File Byte Offset =00000000

BOOTP/RARP Request Retry =00

TFTP/ARP Request Retry =00

Trace Character Buffer Address =00000000

BOOTP/RARP Request Control: Always/When-Needed (A/W)=W

BOOTP/RARP Reply Update Control: Yes/No (Y/N) =Y

PPC1-Bug>

Refer to the Motorola PPCBug firmware manual for further details of the niot boot parameters. The Argument File Name is used at boot time to define the EPICS startup script and the NFS mount command. The Linux host must be configured so that /gem\_sw directory can be mounted by the RTEMS IOC using NFS.

# ADE Directory Structure

The standard directory structure that will be used at Gemini for the real-time software is shown in the figures below.

Figure : ADE Directory Structure

Notes:

* The top-level (ADE root) directory is shown as **/gem\_sw** throughout this document.
* **/gem\_sw** contains the following directories:
* **/epics** - EPICS source code and installed release tree. Contains all EPICS versions in use. Which version is used is indicated by the directory name e.g. **R3.14.12.4.** The version used by a user is defined by the environment variable $GEM\_EPICS\_RELEASE.
* **/etc** - contains the profile file which everyone should source if they want to use the Gemini ADE. It may also contain other system-wide configuration files.
* **/prod**
* **/etc –** Files used by the build system, such as scripts used bythe build server
* **/common –** Common scripts and tools used by the ADE. In particular, subdirectory ***python*** contains python scripts.
* **/Rx.y.z** - Released versions of software checked out from Subversion; built using the named EPICS releases. Which version is used is indicated by the directory name e.g. **prod/R3.14.12.4.**
* **/targetOS** - Cross compilation environments for building target systems. Currently only RTEMS is available.
* **/tools** - external controls applications (not currently used).
* **/work** – Development files and built dependents. Allows sharing of non-production files between IOCs.
* Note that a system in **work** may be using some support modules from **prod** and others from **work**, as well as versions from a developer’s home directory. Which is used is defined in the configuration files for an individual system.
* The format of the EPICS and RTEMS file trees inside ***epics***and ***targetOS*** is the same as distributed and documented by the suppliers.
* All local developed software is kept under either ***prod***or***work.***
* The developer’s home test area is not shown. This is typically their home directory or on a local scratch disk. Note that home directories should be, by default, globally readable.

## work Directory

The work directory tree contains code that is still under development but of a sufficient quality to allow it to be used in other IOC’s applications other than a local test application. The software is located within the tree for the EPICS release with which they were built and with which they can be used.

For a particular EPICS release there will be **support** and **ioc** sub-directories.

### work/<EPICS Release>/support

A set of support modules is stored in directories beneath this directory. The figure below shows an example directory hierarchy within */gem\_sw/work/R3.14.12.4/support* for a support module called astlib. The diagram is expanded for a single support module (astlib); there would be many other, similar hierarchies in */gem\_sw/work/R3.14.12.4/support.*



Figure 3 Contents of /gem\_sw/work/R3.14.12.4/support

#### Make generated directories

In the figure, the **include**, **db**, **dbd**, **data** and **lib** directories are coloured red to mark the fact that they do not contain source files but are produced by the build process. They must not be stored in Subversion**.** Note that not every module will have all the sub-directories shown. In particular, modules which are purely libraries may not have an associated database files in /db,or /dbd.

* **include** - contains the header files for this module, especially those which define interfaces to this module which other applications may want to use.
* **db** – contains any database and template files provided for this module, these will have the “.sch” , “.db” or “.template” file extension.
* **dbd** – contains any database definition files which are unique to this module.
* **lib**/… - objects for host and target architectures and operating systems, determined by the cross-compilation configuration in configure/CONFIG\_SITE.
* **lib/RTEMS-mvme2307** – contains the object library for RTEMS applications running on the Motorola MVME2700 board. If cross-compilation is being done for the MVME6100 target then there would be a similar directory called **lib/RTEMS-beatnik**.
* **lib/linux-x86\_64** – contains the object library for Linux 64-bit applications.

#### configure

The **configure** directory contains the configuration and make rules which will be used when building the application. These would be the standard EPICS rules with some additional, locally defined rules.

The only file that normally requires editing is configure/RELEASE which contains the list of other applications (or more specifically, support modules) on which this application/module depends and the location of these applications. See section 3.7 for more details.

The file configure/CONFIG\_SITE may need to be edited when the module is created, to define the required cross-compilation architectures (see Section 3.8 above).

#### astlibApp

The **astlibApp** is the application directory, containing sub-directories containing the source code files for this support module:

* **src** – contains C/C++ source code, scripts, database definition (.dbd) and sequencer (.st) source code files for this module.
* **Db** – contains any database schematic and symbol files defined for this module. These will have the **“**.sch” and **“**.sym” extensions for use with TDCT. It will usually contain a local TDCT configuration file called tdct.cfg.

### work/<EPICS Release>/ioc

The **ioc** directory hierarchy contains the files that are used operationally for specific IOCs. The figure below shows the directory hierarchy within */gem\_sw/work/R3.14.12.4/ioc/AG/MK* which contains the code to generate the IOC for the A&G (AG) subsystem IOC at Gemini North (MK). This IOC’s full name is therefore AG-MK-IOC.The diagram expands a single IOC instance (MK) but similar hierarchies will exist throughout */gem\_sw/work/R3.14.12.4/ioc.*

Figure Contents of /gem\_sw/work/R3.14.12.4/ioc



#### Make generated directories

In the figure the **db**, **dbd**, **data** and **bin** directories are coloured red to mark the fact that they do not contain source files but are created during the build process. They must not be stored in Subversion.

* **db** – contains the built database (\*.db) files for this IOC.
* **dbd** – contains a single (master) database definition file (.dbd file) for this IOC.
* **data** – contains architecture independent files e.g. configuration data files specific to this IOC.
* **bin/RTEMS-mvme2307** – will contain object file called AG-MK-IOC containing the compiled code for an IOC running RTEMS on an MVME2700 processor. The binary file incorporates the EPICS base software as well as any support module software that this IOC depends upon. This directory also contains the installed startup script (stAG-MK-IOC.boot) file used when booting this IOC.
* **bin/linux-x86\_64** – will contain a file called AG-MK-IOC. This will be an x86\_64 executable file appropriate for running this IOC on a 64 bit Linux target. This directory also contains the installed startup script (stAG-MK-IOC.boot) file used when booting this (soft) IOC.

#### configure

The **configure** directory contains the configuration and make rules which will be used when building the application. These would be the standard EPICS rules with some additional, locally defined rules.

The only file that normally requires editing is configure/RELEASE which contains the list of other applications (or more specifically, support modules) on which this application/module depends and the location of these applications. See section 3.7 for more details.

The file configure/CONFIG\_SITE may need to be edited when the module is created, to define the required cross-compilation architectures (see Section 3.8 above).

#### AG-MK-IOCApp

The **AG-MK-IOCApp** directory includes sub-directories which contain the source files specific to this IOC:

* **src** – The source files on which an IOC depends are normally found in the relevant support modules “src” directory. However, every application under EPICS 3.14 includes a special C++ source file, called <App>Main.cpp, (in this case, AG-MK-IOCMain.cpp). This source file runs the soft IOC shell in the case where the application is run on a target operating system which is not RTEMS.
* **config** - This directory contains configuration data files which are specific to this IOC. This might include site-specific data used by the application, look-up tables and specific hardware configuration data.
* **Db** – contains only a Makefile which defines which databases are to be used by this IOC. The specified database (.db) files will be copied from the relevant module(s) using the module list defined in file configure/RELEASE. When loaded, the EPICS database record names will have a site-specific prefix, defined in a configuration file.

#### iocBoot

The **iocBoot** directory contains subdirectory **iocAG-MK-IOC** which contains the source startup script for this IOC. For this example, it will be called st AG-MK-IOC.src.

## Production (“prod”) Directory

The **prod** directory hierarchy contains software that has been tested and released for production use. The only difference between the structure of the **prod** and **work** directory trees is that the files for each support module and IOC are placed in a release-specific sub-directory, with the same name as the release. For each release the hierarchy is the same as in **work**. This is illustrated in the figures below, for both support and IOC modules.



Figure 5 Structure of support module in /gem\_sw/prod/R3.14.12.4/support



Figure 6 Structure of IOC application in /gem\_sw/prod/R3.14.12.4/ioc

# ADE profile and Environment Variables

The file /gem\_sw/etc/profile file contains Gemini ADE environment definitions and this file must be sourced (usually from the ~/.bashrc file) by anyone using the Gemini ADE. This file will need to be edited for the local site where it is deployed. The environment variables defined include the following

|  |  |
| --- | --- |
| GEM\_ROOT | The base directory of the Gemini ADE tree, here assumed to be /gem-sw. |
| GEM\_IPNUM | The Ethernet address of the current host’s primary Ethernet adaptor. This is derived using information on the local host. |
| GEM\_SITE | The Gemini site (North = "MK" or South = "CP"). This is derived using assumptions about the LAN network addresses at the Gemini sites. |
| GEM\_EPICS\_RELEASE | The EPICS release used by the Gemini ADE. |
| SVN\_ROOT | URL of the Subversion repository location e.g. <http://source.gemini.edu/software>.  Note: throughout this document it is assumed that the Gemini applications area will be located in the Subversion repository under $SVN\_ROOT/gem |
| GEM\_EMAIL\_DOMAIN | The email domain used when sending automatic emails from the build server. Used in the Python script gem-release.py. |
| SVN\_EDITOR | The editor used to enter comments when using SVN. |
| GEM\_SCRIPTS | The location of the main Python ADE scripts (names gem\_\*.py). |
| PYTHONPATH | Defines where imported Python packages can be found. |
| PATH | The executable search path, including RTEMS, EPICS and Python scripts locations. |
| HOST\_ARCH | (EPICS) Host architecture type : linux-x86\_64 for Linux 64-bit. |
| EPICS\_RELEASE | (EPICS) EPICS version (same as $GEM\_EPICS\_RELEASE). |
| EPICS\_HOST\_ARCH | (EPICS) Host architecture type (same as $HOST\_ARCH). |
| EPICS | (EPICS) Root directory location for EPICS version being used. |
| EPICS\_BASE | (EPICS) EPICS Base tree location. |
| EPICS\_EXTENSIONS | (EPICS) EPICS Extensions tree location. |

# Build & Release Scripts Reference

A set of Python scripts should be used to automate interactions with the Subversion repository for software development for the Gemini systems. The source files for the Python scripts are in the scripts/python subdirectory of the makeGemApp module. The Makefile provided installs the scripts into directory $GEM\_ROOT/prod/common/python/GemPySvn.

The scripts are described in the following sections, each with a table which displays the parameters, a description, all the available options for the script, followed by an example.

## gem-start-new-module.py

|  |  |
| --- | --- |
| gem-start-new-module.py [options] <module\_name> | |
| Create a new support or IOC module called <module\_name>, using Gemini ADE templates, import it into the SVN repository and check it out into the current working directory. Options are described below.  The script calls the standard EPICS makeBaseApp.pl script, using Gemini ADE templates to create the new module’s directory structure, including an initial template Makefile in each directory. The user then develops the application by creating TDCT schematic files, C source code, configuration files etc., and modifying the template Makefiles appropriately. | |
| -a AREA --area=AREA | Set the module type to be AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to create an IOC application, in which case <module\_name> is expected to be of the form "Subsystem/Location" e.g. AG/MK.  An additional third part of the name may, optionally, be provided to specify the number for the IOC e.g. AG/MK/01. This would only be necessary if more than one IOC is being used for a particular application at the same site. |
| -n --no\_import | Create a local module but don’t import into Subversion. |
| -h --help | Display help for using the script |

Examples

1. gem-start-new-module.py mysupport

The command creates this repository directory in Subversion:

$SVN\_ROOT/gem/trunk/support/mysupport

and checks out the module into directory ./mysupport in the current working directory.

1. gem-start-new-module.py –i AG/MK

The command creates this repository directory in Subversion:

$SVN\_ROOT/gem/trunk/ioc/AG/MK

The application source will be checked out into directory ./AG/MK/AG-MK-IOCApp.

## gem-checkout-module.py

|  |  |
| --- | --- |
| gem-checkout-module.py [options] <module\_name> | |
| Checks out the specified support or IOC module <module\_name> from the latest trunk (by default) revision in the Subversion repository. Checks out into the current working directory. Options are described below. | |
| -a AREA --area=AREA | Set the module type to be AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to create an IOC application, in which case <module\_name> is expected to be of the form "Subsystem/Location" e.g. AG/MK. |
| -b BRANCH --branch=BRANCH | Checkout the module from the Subversion branch named BRANCH, rather than from the trunk |
| -f --force | Disable warnings, force checkout |
| -h --help | Display help for using the script |

Example

gem-checkout-module.py –i GEMTEST/MK

This command will check out the IOC application GEMTEST/MK from the trunk into the current working directory.

## gem-list-modules.py

|  |  |
| --- | --- |
| gem-list-modules.py [options] [dom\_name] | |
| This script returns a list of all the support modules or IOC collections in the repository. A parameter (dom\_name) may be supplied for ioc modules. Without a parameter it lists all modules in the specified area (support or ioc). | |
| -a AREA --area=AREA | List all modules of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to list IOC modules, in which case <dom\_name> is expected to be either of the form "Subsystem" e.g. AG or "Subsystem/Location" e.g. AG/MK.  Without a parameter it lists all modules in the ioc area |
| -h --help | Display help for using the script |

Example

gem-list-modules.py –i GEMTEST

This command will list all the modules below the IOC top-level application GEMTEST.

## gem-changes-since-release.py

|  |  |
| --- | --- |
| gem-changes-since-release.py [options] <module\_name> | |
| This script checks if a support or IOC module in the Subversion repository has had any changes committed since its last release.  It shows only changes made to the trunk version as well as any releases. | |
| -a AREA --area=AREA | List modules of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to list IOC modules, in which case <module\_name> is expected to be of the form "Subsystem/Location" e.g. AG/MK. |
| -h --help | Display help for using the script |

Example

gem-changes-since-release.py –i GEMTEST/MK

This command will show any changes committed to the Subversion repository for the IOC application GEMTEST/MK, since it was last released.

## gem-logs-since-release.py

|  |  |
| --- | --- |
| gem-logs-since-release.py [options] <module-name> [<earlier-rel>] [<later-rel>] | |
| This script prints all the log messages for module <module-name> in the ioc or support area.  The output can be specified to show only those changes between the revision when <earlier-rel> was done to the revision when <later-rel> was done. If not specified, <earlier-rel> defaults to revision 0 and <later-rel> to the head revision.  If <earlier-rel> is given an invalid value, it will be set to the latest release. | |
| -a AREA --area=AREA | List modules of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to list IOC modules, in which case <module\_name> is expected to be of the form "Subsystem/Location" e.g. AG/MK. |
| -v , --verbose | Print additional log information, including all the files changed in Subversion with each revision. |
| -r, --raw | Print raw text, without colour. If this option is not used, the user name and revision number are displayed using red text. |
| -h --help | Display help for using the script |

Example

1. gem-logs-since-release.py supportmod release1 release2

This command will print all the log messages for the support module called supportmod in SVN from the revision number when release1 was done, to the revision when release2 was done.

1. gem-logs-since-release.py -i AG/MK

No release names are provided, so this command will print all the log messages for the IOC module called AG/MK in SVN since it was created.

## gem-release.py

|  |  |
| --- | --- |
| gem-release.py [options] <module-name> <release\_#> | |
| Release <module-name> at version <release\_#> in the production (prod) ioc or support area.  This script first performs a local test build of the module, and if it succeeds, creates the release in Subversion. It then creates a build script file that is put in a queue for use by the ADE build server, causing it to schedule a checkout and build of the Subversion module release in the production (prod) area.  If the release already exists in prod, no release will be done: a release only can be done in this case by using the –f option. | |
| -a AREA --area=AREA | Release module of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the release module type to be ioc, in which case <module-name> is expected to be of the form "Subsystem/Location" e.g. AG/MK. |
| -b BRANCH --branch=BRANCH | Release the module from the Subversion branch named BRANCH, rather than from the trunk |
| -f, --force | Force a release. If the release already exists in prod, the existing release will be removed and rebuilt. |
| -t, --test\_build | If set, this will skip the test build stage and just schedule a release. |
| -e EPICS\_VERSION,  --epics\_version=EPICS\_VERSION | Use the specified EPICS version for the build rather than the default defined in environment variable GEM\_EPICS\_RELEASE. |
| -m MESSAGE,  --message=MESSAGE | Add a user message to the end of the default Subversion commit message. The message will be '<module\_name>: Released version <release\_#>. <message>’ |
| -h --help | Display help for using the script |

Example

gem-release.py –i GEMTEST/MK Version1

This will release the IOC collection created in the example given for gem-start-new-module above into the directory $SVN\_ROOT/gem/release/GEMTEST/MK/Version1

## gem-start-feature-branch.py

|  |  |
| --- | --- |
| gem-start-feature-branch.py [options] <module\_name> <branch\_name> | |
| This script creates a branch of the current state of the trunk for a support module or IOC application collection. The property gem:synced-from-trunk is added to the branch. The branch is checked out into the current working directory.  If the current working directory is the top directory of a working copy of the trunk (there are .svn files present to allow Subversion to link the current directory with the trunk) then the user is able to update the current working directory so that it is now linked with the new branch.  However, if the trunk has been changed since the trunk was checked out into the current working directory, then switching to the new branch is not possible. | |
| -a AREA --area=AREA | Use modules of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to branch from an IOC module, in which case <module\_name> is expected to be either of the form "Subsystem/Location" e.g. AG/MK. |
| -h --help | Display help for using the script |

Example

gem-start-feature-branch.py –i GEMTEST/MK Branch1

This command will create the directory

$SVN\_ROOT/gem/branches/ioc/GEMTEST/MK/Branch1 from $SVN\_ROOT/gem/trunk/ioc/GEMTEST/MK; and then checks out the branch into “./Branch1”

To see which revision of the trunk was used to create the branch run the commands:

cd Branch1

svn proplist --verbose

## gem-start-bugfix-branch.py

|  |  |
| --- | --- |
| gem-start-bugfix-branch.py [options] <module\_name> <release> <branch\_name> | |
| This script starts a new bugfix branch. This is to be used when a release has been made and we need to patch that release.  The script copies the release called <release> of module <module\_name> into a new branch <branch\_name> and checks it out into the current directory. | |
| -a AREA --area=AREA | Use module of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to branch from an IOC module, in which case <module\_name> is expected to be either of the form "Subsystem/Location" e.g. AG/MK. |
| -h --help | Display help for using the script |

Example

gem-start-bugfix-branch.py -i GEMTEST/MK V1-0 Version1Branch

This command will create the directory $SVN\_ROOT/gem/branches/ioc/GEMTEST/MK/Version1Branch from $SVN\_ROOT/gem/branches/ioc/GEMTEST/MK/V1-0 and then checks out the branch into “./Version1Branch”

## gem-sync-from-trunk.py

|  |  |
| --- | --- |
| gem-sync-from-trunk.py [options] | |
| This script will synchronise a local working copy of a feature branch, for a support module or IOC application, with the latest version on the trunk. For a project checked out from trunk simply run the command “svn status”.  No changes are made to the repository as a result of running this script. The only changes will be to files in the local working copy of the branch. All changes in the local working copy should be checked and any conflicts resolved. These changes then need to be committed back to the feature branch. | |
| -h --help | Display help for using the script |

Example

Having run gem-start-feature-branch.py to create branch “Branch1” we want to pick up changes made by others to the trunk

cd Branch1

gem-sync-from-trunk.py

To get a listing of any conflicts introduced by the update run the command

svn status

## gem-list-branches.py

|  |  |
| --- | --- |
| gem-list-branches.py [options] <module\_name> | |
| This script returns a list of branches of a named support module or IOC collection. | |
| -a AREA --area=AREA | List branches of modules of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to list branches of IOC modules, in which case <module\_name> is expected to be either of the form "Subsystem/Location" e.g. AG/MK. |
| -h --help | Display help for using the script |

## gem-vendor-import.py

|  |  |
| --- | --- |
| gem-vendor-import.py [options] <source> <module> <version> | |
| This script imports a support module or IOC application provided by a vendor into the Subversion repository.  The script imports the code from <source> to a new (vendor) module in Subversion at $SVN\_ROOT/gem/vendor/<area>/<module>/<version>.  It also copies the code to the trunk and checks it out into the current directory. | |
| -a AREA --area=AREA | Import modules of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to import an IOC module, in which case <module> is expected to be either of the form "Subsystem/Location" e.g. AG/MK. |
| -h --help | Display help for using the script |

Example

gem-vendor-import.py sources Vendor1 Version1

This command imports the contents of the directory ./sources into $SVN\_ROOT/gem/vendor/support/Vendor1/Current and copies it to $SVN\_ROOT/gem/vendor/support/Vendor1/Version1, as well as $SVN\_ROOT/gem/trunk/support/Vendor1.

## gem-vendor-update.py

|  |  |
| --- | --- |
| gem-vendor-update.py [options] <source> <module> <old> <new> | |
| This script creates a new version of a vendor support module or IOC application by merging an existing version label with a new version.  This script updates vendor <module> from tag <old> to tag <new> using the code from <source>.  The updated vendor module is imported into: $SVN\_ROOT/gem/vendor/support/<module>/current  which contains a history of all the releases made by the vendor. The code is then copied into:  $SVN\_ROOT/gem/vendor/support/<module>/<new>.  <old> must be the tag used for the previous import into "current" i.e. it would have been tag <new> the last time this script was run.  The user must perform an additional step to merge the new revision onto the trunk as the trunk may have been edited since delivery from the vendor. Instructions are displayed when the script is run. | |
| -a AREA --area=AREA | Update a module of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to update an IOC module, in which case <module> is expected to be either of the form "Subsystem/Location" e.g. AG/MK. |
| -h --help | Display help for using the script |

Example

gem-vendor-update.py sources Vendor1 Version1 Version2

This command imports the contents of ./sources into $SVN\_ROOT/gem/vendor/support/Vendor1/Version2 and $SVN\_ROOT/gem/vendor/support/Vendor1/Current.

To update the trunk version do the following:

svn checkout $SVN\_ROOT/gem/trunk/support/Vendor1

cd Vendor1

svn merge $SVN\_ROOT/gem/vendor/support/Vendor1/Version1 $SVN\_ROOT/gem/vendor/support/Vendor1/Version2

svn status

Fix any conflicts and commit the changes.

## gem-get-vendor-current.py

|  |  |
| --- | --- |
| gem-get-vendor-current.py [options] <module> | |
| This script is used to determine the last tag of <module> which was imported into $SVN\_ROOT/gem/vendor/support/<module>/current.  This tag should be used as the input argument <old> to gem-vendor-update.py. | |
| -a AREA --area=AREA | Get tag for module of type AREA, where AREA is support or ioc.  The default is support. |
| -i --ioc | Set the module type to be ioc, to get the tag for an IOC module, in which case <module> is expected to be either of the form "Subsystem/Location" e.g. AG/MK. |
| -h --help | Display help for using the script |

This script is used to determine the last tag of module which was imported into

$SVN\_ROOT/gem/vendor/support/module/current.