

# EPOCHS Instruction Manual

Kenneth M. Hopkinson  
Copyright © 2003 Cornell University

July 9, 2003

## **Overview**

EPOCHS, the **E**lectric **P**ower and **C**ommunication **S**ynchronizing **S**imulator combines UC Berkeley's Network Simulator 2 (NS2) with GE's PSLF electromechanical transient simulation system and HVDC Manitoba's PSCAD electromechanical simulator together into one federated simulation environment. This is a powerful environment allowing the power and communication communities to examine how interdependencies between communication and power equipment interact under complex scenarios. This guide is designed to give an overview of the use of the EPOCHS environment. The power and communication environment has been created as research software. Every effort has been made to make a bug-free and user-friendly environment, but the emphasis of our small development team is on functionality that can be used for quick development that leads to definite research results. Undoubtedly, there are bugs and non-intuitive steps that have crept in along the way. We hope that this is useful for you and would like to hear from you if you have any questions, comments, or would like to add to our development effort. For more information about the structure of EPOCHS itself is the Winter Simulation Conference paper listed in [1]. This is also a good paper to cite when describing work that makes use of the EPOCHS environment in research papers.

## **Installation**

### **Required Software**

- Microsoft Visual C++ 6.0: We are using the native compile method for NS2. This option uses Visual C++ to create a binary file that can run directly on the Windows operating system. Visual C++ 7.0, aka Visual C++ .NET, restructures a number of compile switches and directory contents. EPOCHS should be able to compile with some modification with Visual C++ 7.0, but we have not attempted to do so.
- PSLF: EPOCHS makes use of PSLF for electromechanical transient situations. PSLF is not required for electromagnetic transient scenarios.
- PSCAD/EMTDC: EPOCHS makes use of PSCAD/EMTDC for electromagnetic transient situations. PSLF is not required for electromechanical transient scenarios. If you are using PSCAD/EMTDC then you should also have the g++ distribution available from their web site installed on your system.

## External Software Included in the EPOCHS Distribution

- NS2: EPOCHS depends on NS2 for simulated network communication. EPOCHS currently requires NS2 version 2.1b9a. The agentsimv9a directory is a modified version of the distribution that has been augmented with the requisite EPOCHS code. You can acquire the original NS2 source code from <http://www.isi.edu/nsnam/ns/>.
- SCTP: We have patched the NS2 distribution to add SCTP support using version 3.2 of the University of Delaware's SCTP module. The SCTP module has been modified to run on the Windows operating system and to work with NS2 version 2.1b9a. You can download the original SCTP patch from <http://pel.cis.udel.edu/>.

## Installing EPOCHS

The EPOCHS distribution is designed to be unzipped into the root directory of your C drive. It will create a directory entitled agentsimv9a containing the NS2 distribution modified to include the SCTP 3.2 release and the EPOCHS system. A directory called "epc" is also created. It contains EPOCHS-specific code in the "code" subdirectory and executables in the "swap" subdirectory. You will also find the instructions that you are reading now as well as the distribution's license agreement. If Visual C++ is installed in a different location than the one specified in the NS2 makefiles then you will need to make the appropriate modifications. If you wish to move the location for your EPOCHS compilation execution then you can do so at this time by making changes as described in the "Changing EPOCHS Options" section.

## Changing EPOCHS Options

EPOCHS has a number of compile options that are located in epochs\_conf.h. These #defines are off when they are preceded by a "/" C++ comment and otherwise are on. They are:

- Windows: Almost all EPOCHS code is written in ANSI C++, but there are a few operating system-specific functions that differ between Windows and Unix. If this option is on then the code will compile for the Windows OS. If it is off then Unix is assumed.
- PSCAD: Turn this on to compile for PSCAD operation. Please note that only one of the two power simulators can be turned on at once in the current EPOCHS system.
- PSLF: Turn this on for PSLF operation. Please note that only one of the two power simulators can be turned on at once in the current EPOCHS system.
- TRACK\_AGENT\_INTERACTIONS: A record of the power and communication messages is kept if this flag is turned on.
- WRITE\_CHECKPOINTS: This is reserved for future use. It should be turned off in the current release.
- READ\_CHECKPOINTS: This is reserved for future use. It should be turned off in the current release.
- TEMP\_PATH: This is the directory where EPOCHS simulations will be executed.

- CHOOSINGHQ: The name of the file to use for the agenthq's choosing file. This file is used to ensure that only one program writes to a log file at a time.
- CHOOSINGPWR: The name of the file to use for the power simulator's choosing file. This file is used to ensure that only one program writes to a log file at a time.
- NUMBERHQ: The name of the file to use for the agenthq's number file. This file is used to ensure that only one program writes to a log file at a time.
- NUMBERPWR: The name of the file to use for the power simulator's number file. This file is used to ensure that only one program writes to a log file at a time.
- ASCII\_TRUE: This is the ASCII string used to represent "true".
- ASCII\_FALSE: This is the ASCII string used to represent "false".
- LOG\_FILE: The file used to log power and communication messages.
- MSG\_IN\_FILE: The input power system exchange file suffix.
- MSG\_OUT\_FILE: The output power system exchange file suffix.
- CONTROL\_LOG: Reserved for future use. This should be turned off in the current release.

## ***Running Cases***

The EPOCHS distribution comes with a number of sample cases. Running a case involves starting both the power simulator and network communication simulator inside the EPOCHS' swap directory.

Starting a PSCAD/EMTDC scenario involves loading the case library, such as backup3.psl, followed by loading and then running the PSCAD/EMTDC case, such as backup\_agent\_0314exp1.psc.

To begin a PSLF case, open its .sav load flow component using getf followed by getting its .dyd dynamic data file in the psds PSLF section. There are a number of values that must be set when running the dynamic case:

- Next Pause Time: Set to the simulation end time
- Steps/plot (file): Set to 1 to ensure that PSLF will keep its timing steady
- Steps/display: Set to 1 to ensure that PSLF will keep its timing steady
- Maximum Time: Should equal the "Next Pause Time" value
- Step width, sec: Time in seconds between EPOCHS rounds. This must equal the time set in the NS case script.
- Unsteady rate tol: Set to 0.0000 to ensure that PSLF will keep its timing steady

The network simulator should be run in the swap directory using "ns script.tcl" where script is the name of the case script. Each EPOCHS case has a subdirectory where its script generator is located. PSLF generators are compiled using Visual C++. Background traffic settings and case input files are found in ieee\_driver.cpp. All other settings are found in the "generate" function inside convert\_ieee.cpp. PSCAD/EMTDC generators are compiled using g++. Simulation variables are set inside generator.cc. The cases are:

## **PSCAD Test**

This is a PSCAD-based case is centered on the use of backup protection agents that use communication to augment their capabilities.

## **SPS**

This PSLF-based case is a simplified version of the SPS Communication case. The case makes use of a special protection system that uses information from the system generators to detect pending transient instability and uses load shedding and generator rejection to maintain the system. This SPS has been designed to protect a modified version of the IEEE 50 generator system. No information is ever transmitted over the communication network here. The SPS can magically detect the instantaneous state of any bus in the system.

## **SPS Communication**

This PSLF-based case is an advanced version of the SPS case. The case makes use of a special protection system that uses information from the system generators to detect pending transient instability and uses load shedding and generator rejection to maintain the system. This SPS has been designed to protect a modified version of the IEEE 50 generator system. Information is transmitted over the communication network here and both losses and long transmission delays can occur. The SPS can only directly detect the state of its own bus in this system.

## **Bilateral**

This PSLF-based case centers on simulated load-following (aka load frequency regulation) using bilateral contracts. That is, variable-power contracts between generators and loads in different control areas.

## **Directional Relaying**

Power system engineers sometimes run scenarios to see which relays will trip in a given situation. The goal of these particular tests is to see which relays trip rather than when they trip or the resulting dynamics in the power system. This PSLF-based case allows a user to take any power system in the IEEE Common Data Format and create a scenario using directional relays. No dynamic data is required in this case. Directional relays are created at each bus in the system. It is assumed that each relay can detect a fault up to N lines away from it. N has a default value of 3, but can be set as desired. Blocking signals are sent from one side of a line to the other as long as a fault is not detected. The background traffic module can be used with this case to get a sense of how varying levels of background traffic affect the proper operation of the relays in the system.

## ***Creating New Cases***

The EPOCHS system has a well-defined procedure for adding new simulation cases.

- Add a new entries into the `epochs_agent.h` `agenthq_agent_types` enumeration. You should add one new entry per added agent type. The position in the list yields the identification number for a given agent type. For example, the BILATERAL agent type is the fourth entry in the `agenthq_agent_types` enumeration so its EPOCHS id is 3. The id is not 4 because enumerations begin count at 0 by default.
- Create a new agent class for each new agent type. You should add a new entry into the `create_epochs_entry` section of each new agent specifying its corresponding class.
- EPOCHS agents are children of the `EPOCHSAgent` class. Agents should have a “request” method where power system information is requested at the beginning of each round. The “action” method is where the actual power system/communication/computation takes place each round. Agent communication is received in the “`recv_comm_msg`” method. Power system communication is similarly received in the “`recv_power_msg`” method. The agent can be structured as desired given these restrictions. You can use the agents provided as templates for your own work.
- Create a script generation directory similar to those corresponding the sample EPOCHS cases. Variables specific to individual agents should be placed in the NS2 tcl script generated.
- Create the power system portion of your test case. You can use the sample test cases as a template for this part of agent creation process.
  - PSCAD/EMTDC: PSCAD test cases require that you use the `control.h/control.c` and `pscadtcp.h/pscadtcp.c` files in order to interface with the EPOCHS system. You should customize these files to your system’s structure and to reflect the information that should be passed between PSCAD and the EPOCHS agents.
  - PSLF: PSLF cases do not need to be customized to each individual scenario as is the case with PSCAD. You should ensure that the `EPCL test23_debug.p` file includes the power system commands that you wish to use, though.
- Finally, place the case files in your swap directory. You should be ready to run and debug your cases at this time.

## ***Using the Background Traffic Module***

The background traffic module generates communication traffic that is likely to appear in a real Utility Intranet. Each traffic component can be set to whatever frequency is desired. The components descriptions and suggested traffic frequencies consist of:

- Supervisory Control and Data Acquisition (SCADA) Data  
(Suggest One Complete Area Scan / Second)
  - Injections (MW, MVAR)
  - Flows (Real, Reactive)
  - Voltage
  - Breaker Status (In or Out of Service)

- Power Quality Data  
(Suggest One Reading From Each Industrial and Some Commercial Centers Per Second)
- Power Trading Traffic  
Customers can choose to buy their power under current market conditions that will be updated (suggest every 5 minutes) based on the current nodal market price of power. All nodal pricing will need to be available together in one place for some market places. Most locations will require selective data only.
- Routine Internal Traffic  
Design/Blueprint information, e-mail, and other routine exchanges will likely occur on the intranet. These files could be reasonably large, but these transfers will occur much less frequently than occurs on the Internet since it is strictly on an internal basis. Exchanges are likely to occur between engineering offices and power plants, power plants to other power plants, ISO's to engineering offices, etc.  
(Suggest 100,000 byte transfer size with a probability of 0.002 per area per round, but this is highly dependent on your expected network characteristics).
- Office Substation Traffic  
Signals from offices to substations
  - ISO Control Center to Substations: Typically consists of supervisory control signals telling buses to take actions
  - Engineering Office to Substations: Typically asking to change settings from one value to another

Note that Routine Internal Traffic and Office substation traffic are unlikely to take place at the same time.  
(Suggested packet size of 20 bytes and probability of 0.002 per bus per round. Your actual values will depend on your expected power system operating characteristics.)
- Event Notification Traffic  
When an event occurs, data will be sent from event/fault recorders. An event might be a lightning strike followed by a set of circuit breaker trips in response. Sampled waveforms of voltages and currents at 5 KHz per second are possible. This data can be quite large when compared with many of the other types of information that are passed around the system. These files will be sent after the fault has occurred meaning that they do not interfere with the current situation. However, if a fault occurs and then another follows it then interference could occur. Similarly, if a line is faulted and then an auto-recloser attempts to reconnect then this could cause a second fault.  
(This event only occurs when explicitly triggered. A representative transfer size would be  $5,000 \text{ Hz} * 1 \text{ 16-bit Word/Cycle} * 20 \text{ Seconds} * 12 \text{ Channels} = 2.3 \text{ MB of data}$ )

Each EPOCHS scenario has the option of using the background traffic module. This is typically done by checking the backgroundTraffic flag at time the system is initialized. A value of 1 indicates that the module should be used and a 0 indicates that it should not. The background object should be activated once per round in the action phase. Each of

the sample cases incorporated in the EPOCHS distribution allow the user to make use of background traffic. You may view their source code for more information.

## ***Conclusion***

This guide is a short reference describing how to get up and running with the EPOCHS system. We hope that you have found it helpful. If you have any questions or comments then please send a message to Ken Hopkinson [hopkik@cs.cornell.edu](mailto:hopkik@cs.cornell.edu), Renan Giovanini [rg228@cornell.edu](mailto:rg228@cornell.edu), or Xiaoru Wang [xw44@cornell.edu](mailto:xw44@cornell.edu) and we will do our best to help.

## ***References***

[1] Hopkinson, K.M.; Giovanini, R.; Wang, X.; Birman, K.P.; Thorp, J.S.; Coury, D.V., EPOCHS: Integrated Commercial Off-the-Shelf Software for Agent-based Electric Power and Communication Simulation. 2003 Winter Simulation Conference. 7-10 of December 2003, New Orleans, USA.