Software

IDC DOCUMENTATION

Global Association (GA) Subsystem

Approved for public release;
distribution unlimited

Notice
This document was published May 2001 by the Monitoring Systems Operation of Science Applications International Corporation (SAIC) as part of the International Data Centre (IDC) Documentation. Every effort was made to ensure that the information in this document was accurate at the time of publication. However, information is subject to change.

Contributors
Ronan Le Bras, Science Applications International Corporation

Trademarks
ORACLE is a registered trademark of Oracle Corporation.
SAIC is a trademark of Science Applications International Corporation.
Solaris is a registered trademark of Sun Microsystems.
SPARC is a registered trademark of Sun Microsystems.
UltraSPARC is a registered trademark of Sun Microsystems.
Sun is a registered trademark of Sun Microsystems.
UNIX is a registered trademark of UNIX System Labs, Inc.

Ordering Information
The ordering number for this document is SAIC-01/3009.

This document is cited within other IDC documents as [IDC7.1.4]

Global Association (GA) Subsystem

CONTENTS

About this Document i

- **PURPOSE** ii
- **SCOPE** ii
- **AUDIENCE** ii
- **RELATED INFORMATION** iii
- **USING THIS DOCUMENT** iii

Conventions iv

Chapter 1: Overview 1

- **INTRODUCTION** 2
- **FUNCTIONALITY** 6
- **IDENTIFICATION** 6
- **STATUS OF DEVELOPMENT** 6
- **BACKGROUND AND HISTORY** 7
- **OPERATING ENVIRONMENT** 7

Hardware 7

Commercial-Off-The-Shelf Software 7

Chapter 2: Architectural Design 9

- **CONCEPTUAL DESIGN** 10
- **DESIGN DECISIONS** 14

Programming Language 14

Global Libraries 14

Database 14

Interprocess Communication (IPC) 14

Filesystem 15

Design Model 15

Database Schema Overview 16
Database Entity-relationship Diagram 18
■ **FUNCTIONAL DESCRIPTION** 18
Building the Knowledge Base 21
Visualizing the Knowledge Base 21
Generating New Automatic Events 21
Resolving Conflicts 22
Setting Up Arrival Tags 22
■ **INTERFACE DESIGN** 22
Interface with Other IDC Systems 22
Interface with External Users 23
Interface with Operators 23
Chapter 3: Detailed Design of GAassoc 25
■ **DATA FLOW MODEL** 26
■ **PROCESSING UNITS** 28
Extract Arrival List 30
Read Command-line Parameters 31
Access Knowledge Base 32
Restrict Phase List 34
Associate Arrivals 35
Extract Large Events 44
Eliminate Redundant Events 47
Locate and Confirm Preliminary Event Hypotheses 49
Resolve Conflicts 51
Write Event Hypotheses to Database 52
■ **DATABASE DESCRIPTION** 53
Database Design 53
Database Schema 54
Chapter 4: Detailed Design of GAconflict 57
■ **DATA FLOW MODEL** 58
■ **PROCESSING UNITS** 61
Extract Arrival List 63

Read Command-line Parameters 64
Read Event Information 64
Locate and Confirm Preliminary Event Hypotheses 65
Predict Defining Phases 66
Resolve Conflicts 68
Predict Nondefining Phases 68
Check Consistency 69
Write Event Hypotheses to Database 71
■ **DATABASE DESCRIPTION** 71
Database Design 71
Database Schema 72

Chapter 5: Detailed Design of GA_DBI 75
■ **DATA FLOW MODEL** 76
■ **PROCESSING UNITS** 77
Tag Auxiliary Arrivals 77
Tag Hydroacoustic Arrivals 78
■ **DATABASE DESCRIPTION** 79
Database Design 79
Database Schema 79

Chapter 6: Detailed Design of GAcons 81
■ **DATA FLOW MODEL** 82
■ **PROCESSING UNITS** 83
Build Static Grid 83
Build Grid Files 85
■ **DATABASE DESCRIPTION** 93
Database Design 93
Database Schema 93

Chapter 7: Detailed Design of GAgrid 95
■ **DATA FLOW MODEL** 96
■ **PROCESSING UNITS** 97
Read and Parse Grid File 97

Display Grid Information 98

■ **DATABASE DESCRIPTION** 99

References 101

Glossary G1

Index I1

Global Association (GA) Subsystem

FIGURES

FIGURE 1.	IDC SOFTWARE CONFIGURATION HIERARCHY	3
FIGURE 2.	RELATIONSHIP OF GA TO OTHER SOFTWARE UNITS OF AUTOMATIC PROCESSING CSCI	5
FIGURE 3.	HIGH-LEVEL PROCESSING FLOW SHOWING INTERACTIONS AMONG DIFFERENT GA PROCESSES	12
FIGURE 4.	GA DATABASE TABLE RELATIONSHIPS	19
FIGURE 5.	GA FUNCTIONAL DESIGN	20
FIGURE 6.	GAASSOC DATA FLOW	27
FIGURE 7.	RELATIONSHIPS AMONG GAASSOC DATA STRUCTURES	38
FIGURE 8.	EXTRACT LARGE EVENTS PROCESSING SEQUENCE	46
FIGURE 9.	DOUBLE-LINK RELATIONSHIP BETWEEN DRIVER STRUCTURES AND ARRIVAL_INF STRUCTURES	48
FIGURE 10.	GACONFLICT DATA FLOW	59
FIGURE 11.	INPUT AND OUTPUT BULLETIN TABLES IN GAASSOC AND GACONFLICT	72
FIGURE 12.	GA_DBI DATA FLOW	76
FIGURE 13.	GACONS DATA FLOW	82
FIGURE 14.	PROPAGATION KNOWLEDGE BASE GRID FILE STRUCTURE	87
FIGURE 15.	BEAM POINT RECORD STRUCTURE WITHIN GRID FILES	88
FIGURE 16.	STATION RECORD STRUCTURE	88
FIGURE 17.	RELATIONSHIPS AMONG GACONS DATA STRUCTURES	92
FIGURE 18.	GAGRID DATA FLOW	97

Global Association (GA) Subsystem

TABLES

TABLE I:	DATA FLOW SYMBOLS	v
TABLE II:	ENTITY-RELATIONSHIP SYMBOLS	vi
TABLE III:	TYPOGRAPHICAL CONVENTIONS	vi
TABLE IV:	DATA STRUCTURE AND POINTER CONVENTIONS	vii
TABLE 1:	STANDARD PRODUCTS AVAILABLE THROUGH GA	11
TABLE 2:	DATABASE TABLES USED BY GA	16
TABLE 3:	PROCESSING UNITS AND CORRESPONDING C FUNCTIONS	28
TABLE 4:	PRIMARY DATA CONTENT IN STRUCTURES USED BY GA	37
TABLE 5:	DRIVER STRUCTURE	39
TABLE 6:	COR_STA STRUCTURE	41
TABLE 7:	STA_AR STRUCTURE	42
TABLE 8:	ARRIVAL_INF STRUCTURE	42
TABLE 9:	DR_LIST STRUCTURE	49
TABLE 10:	GAASSOC DATABASE USAGE	54
TABLE 11:	PROCESSING UNITS AND CORRESPONDING C FUNCTIONS	60
TABLE 12:	PRED_TRIPLET STRUCTURE	67
TABLE 13:	GACONFLICT DATABASE USAGE	73
TABLE 14:	GA_DBI DATABASE USAGE	80
TABLE 15:	GRID_PT STRUCTURE	84
TABLE 16:	BEAM_PT STRUCTURE	90
TABLE 17:	FIRST_STA STRUCTURE	90
TABLE 18:	STAPT STRUCTURE	90
TABLE 19:	PHAS_INF STRUCTURE	91
TABLE 20:	GACONS DATABASE USAGE	94

About this Document

This chapter describes the organization and content of the document and includes the following topics:

- Purpose
- Scope
- Audience
- Related Information
- Using this Document

About this Document

PURPOSE

This document describes the design of the Global Association (GA) Subsystem software of the International Data Centre (IDC). The software is part of the Network Processing Computer Software Component (CSC) of the Automatic Processing Computer Software Configuration Item (CSCI). This document provides a basis for implementing, supporting, and testing the software.

SCOPE

The software is identified as follows:

Title: Global Association Subsystem

Abbreviation: GA

This document describes the architectural and detailed design of the software including its functionality, components, data structures, high-level interfaces, method of execution, and underlying hardware. This information is modeled on the Data Item Description for Software Design Descriptions [DOD94a].

AUDIENCE

This document is intended for all engineering and management staff concerned with the design of all IDC software in general and of GA in particular. The detailed descriptions are intended for programmers who will be developing, testing, or maintaining GA.

RELATED INFORMATION

The following documents complement this document:

- *Global Association (GA) Subsystem Software User Manual* [IDC6.5.12]
- *IDC Processing of Seismic, Hydroacoustic and Infrasonic Data* [IDC5.2.1]
- *Database Schema* [IDC5.1.1Rev2]
- *Configuration of PIDC Databases* [IDC5.1.3Rev0.1]

See "References" on page 101 for a list of documents that supplement this document. The following UNIX manual (man) pages apply to the existing GA software:

- *GA*
- *GAassoc*
- *GAconflict*
- *GA_DBI*
- *GAcons*
- *GAgrid*

USING THIS DOCUMENT

This document is part of the overall documentation architecture for the IDC. It is part of the Software category, which describes the design of the software. This document is organized as follows:

- Chapter 1: Overview

  This chapter provides a high-level view of GA, including its functionality, components, background, status of development, and current operating environment.

- Chapter 2: Architectural Design

  This chapter describes the architectural design of GA, including its conceptual design, design decisions, functions, and interface design.

- Chapter 3: Detailed Design of GAassoc

  This chapter describes the detailed design of *GAassoc* including its data flow, software units, and database design.

- Chapter 4: Detailed Design of GAconflict

  This chapter describes the detailed design of *GAconflict* including its data flow, software units, and database design.

- Chapter 5: Detailed Design of GA_DBI

  This chapter describes the detailed design of *GA_DBI* including its data flow, software units, and database design.

- Chapter 6: Detailed Design of GAcons

  This chapter describes the detailed design of *GAcons* including its data flow, software units, and database design.

- Chapter 7: Detailed Design of GAgrid

  This chapter describes the detailed design of *GAgrid* including its data flow and software units.

- References

  This section lists the sources cited in this document.

- Glossary

  This section defines the terms, abbreviations, and acronyms used in this document.

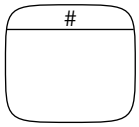
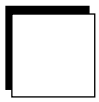
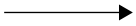
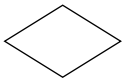
- Index

  This section lists topics and features provided in the document along with page numbers for reference.

Conventions

This document uses a variety of conventions, which are described in the following tables. Table I shows the conventions for data flow diagrams. Table II shows the conventions for entity-relationship diagrams. Table III lists typographical conventions. Table IV shows conventions for the relationships between different data structures within GA.

TABLE I: DATA FLOW SYMBOLS

Description	Symbol[1]
process	
external source or sink of data (left)	
data store (left) duplicated data store (right) M = memory store T = tape store D = disk store Db = database store MS = mass store	
control flow	
data flow	
decision	

1. Symbols in this table are based on Gane-Sarson conventions [Gan79].

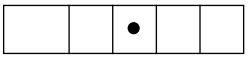
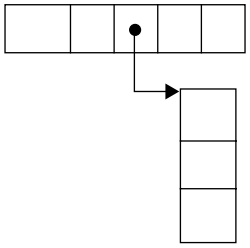
TABLE II: ENTITY-RELATIONSHIP SYMBOLS

Description	Symbol
One **A** maps to one **B**.	A ←——→ B
One **A** maps to zero or one **B.**	A ←——○→ B
One **A** maps to many **B**s.	A ←——→→ B
One **A** maps to zero or many **B**s.	A ←——○→→ B
database table	**tablename** *primary key* *foreign key* *attribute 1* *attribute 2* . . . *attribute n*

TABLE III: TYPOGRAPHICAL CONVENTIONS

Element	Font	Example
database tables	**bold**	**affiliation**
database table and attribute, when written in the dot notation		**assoc.***belief*
database attributes	*italics*	*sta*
processes, software units, and libraries		*GAassoc*
titles of documents		*Configuration of PIDC Databases*
computer code and output	`courier`	`GAdepth_build:`
filenames, directories, and websites		`slowamp.P`
C structures		`Phas_Inf`

TABLE IV: DATA STRUCTURE AND POINTER CONVENTIONS

Description	Symbol
A set of rectangles represents a data structure. Each rectangle represents an element of the structure. These elements may be data or pointers. Data elements are indicated by an empty rectangle. Pointers are indicated by a dot in the center of the rectangle.	
An arrow originating at a dot and pointing to the start of a data structure represents a pointer to the address of another data structure.	
A circle with an "X" represents a null terminator, which terminates a linked list of data structures.	

Chapter 1: Overview

This chapter provides a general overview of the GA software and includes the following topics:

- Introduction
- Functionality
- Identification
- Status of Development
- Background and History
- Operating Environment

Chapter 1: Overview

INTRODUCTION

The software of the IDC acquires time-series and radionuclide data from stations of the International Monitoring System (IMS) and other locations. These data are passed through a number of automatic and interactive analysis stages, which culminate in the estimation of location and in the origin time of events (earthquakes, volcanic eruptions, and so on) in the earth, including its oceans and atmosphere. The results of the analysis are distributed to States Parties and other users by various means. Approximately one million lines of developmental software are spread across six CSCIs of the software architecture. One additional CSCI is devoted to run-time data of the software. Figure 1 shows the logical organization of the IDC software. The Automatic Processing CSCI distributes data through the following CSCs:

- Station Processing

  This software scans data from individual time-series stations for characteristic changes in the waveforms (detection of onsets) and characterizes such onsets (feature extraction). The software then classifies the detections as arrivals in terms of phase type.

- Network Processing

  This software combines arrivals from several stations originating from one event and infers the location and time of its origin.

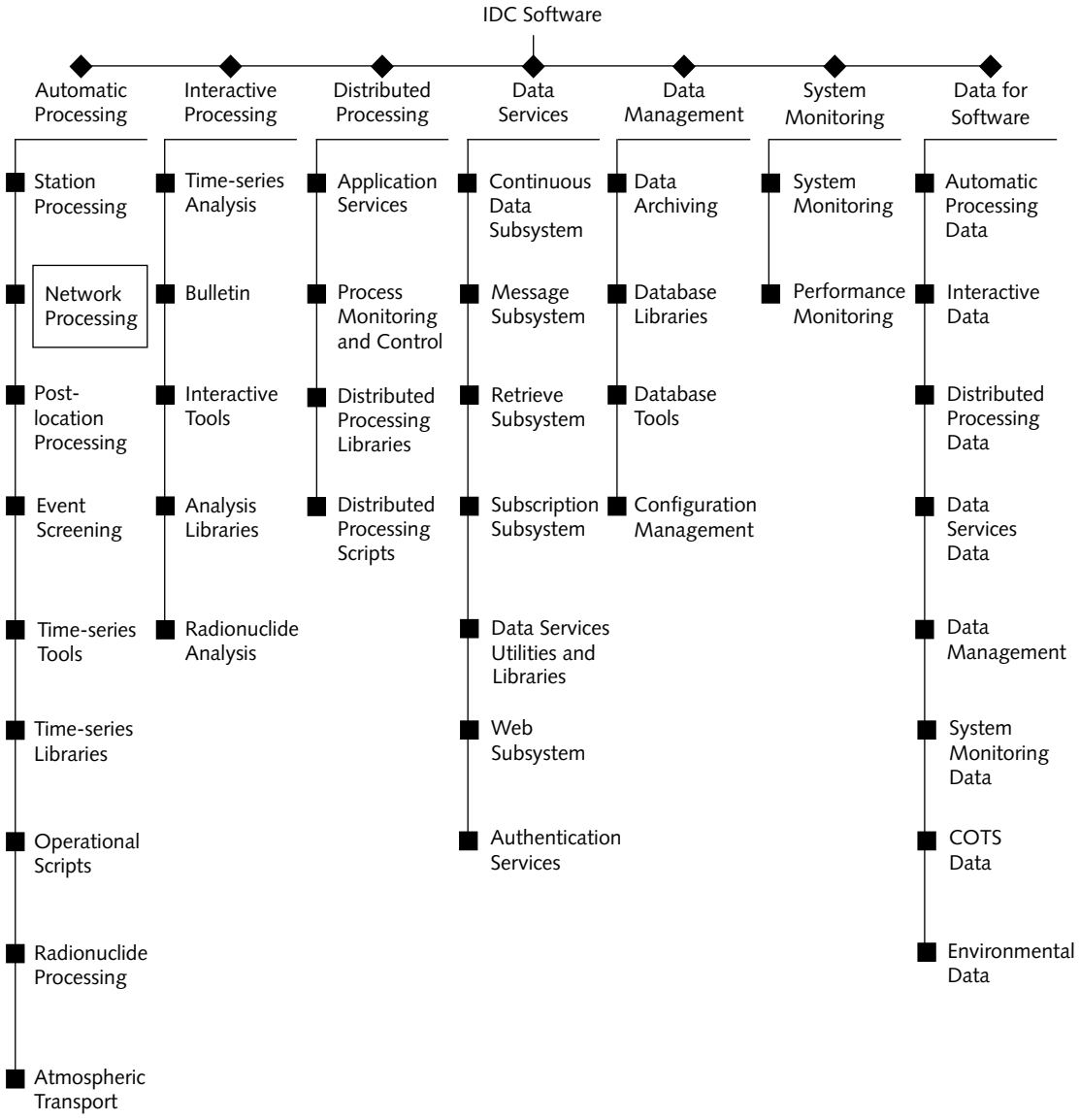


FIGURE 1. IDC SOFTWARE CONFIGURATION HIERARCHY

- Post-location Processing

  This software computes various magnitude estimates and selects data to be retrieved from auxiliary stations.

- Event Screening

  This software extracts a number of parameters that characterize an event; then a default subset of the calculated Event Characterization Parameters eliminates the events that are clearly not explosions.

- Time-series Tools

  This software includes various utilities for the Seismic, Hydroacoustic, and Infrasonic (S/H/I) processing system.

- Time-series Libraries

  This software includes shared libraries to which several modules of the S/H/I processing system are linked.

- Operational Scripts

  This software provides miscellaneous functionality to enable automatic processing to function as a system.

- Radionuclide Processing

  This software includes the automated analysis, categorization, and flagging processes for radionuclide data.

- Atmospheric Transport

  This software includes the forward and backward modeling of the transport of particulates by atmospheric movements.

Figure 2 shows the relationship of GA to components of the Automatic Processing CSCI. This figure indicates that GA fulfills the following roles: It associates the arrivals produced by Station Processing to form event hypotheses. Detections from the network of stations are grouped together into association sets, which define distinct events. These events are located, and their magnitudes are estimated. Figure 2 shows the dynamics of the processing, but does not show the static tables (**site, siteaux, affiliation,** and so on) or waveform data files used by the processes. Station Processing includes the programs *DFX*, which performs detection and fea-

ture extraction, *StaPro*, which performs initial phase identification, and *HAE* (Hydroacoustic Azimuth Estimation), which determines the azimuth for multi-site hydroacoustic stations.

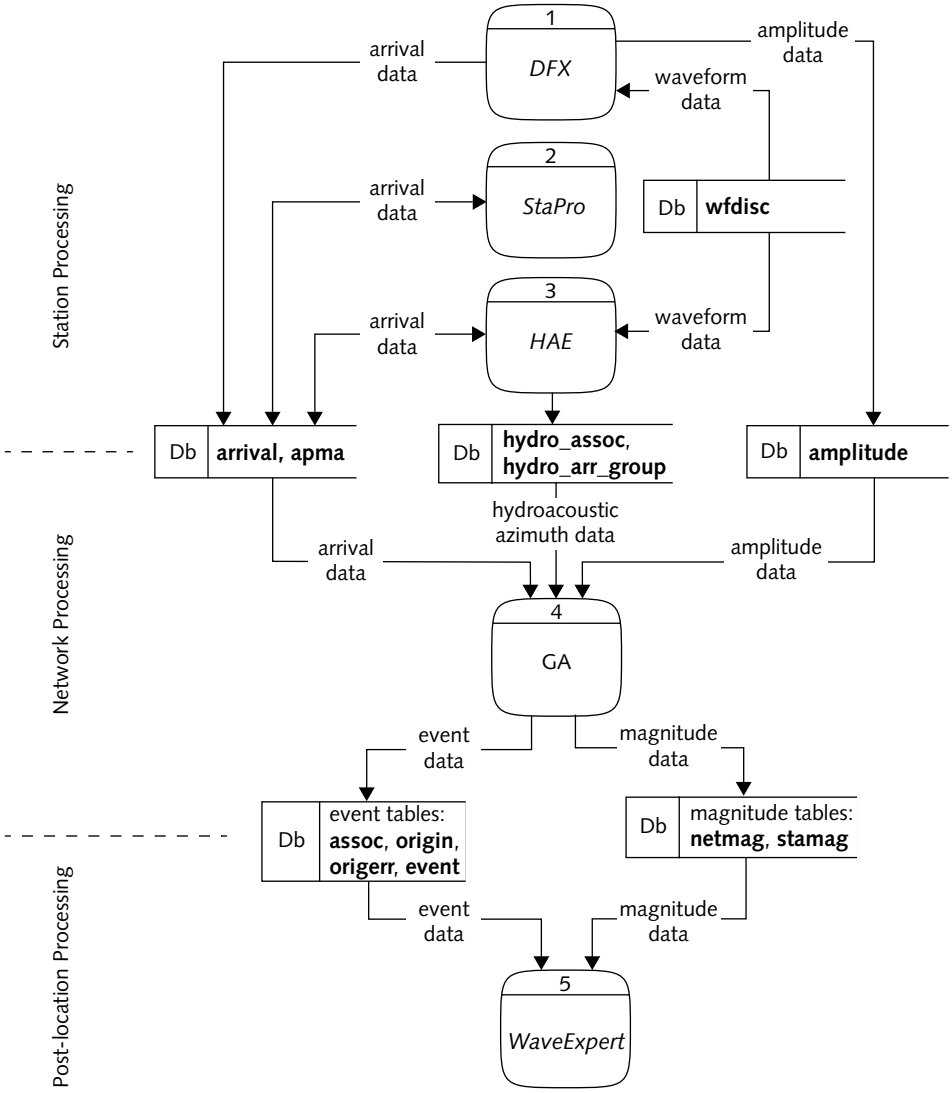


FIGURE 2. RELATIONSHIP OF GA TO OTHER SOFTWARE UNITS OF AUTOMATIC PROCESSING CSCI

FUNCTIONALITY

GA is the process in the automatic pipeline that forms event hypotheses. GA reads arrival and amplitude data for a time interval and forms sets of associations using an exhaustive search algorithm. These association sets define the events, which then are located and have their magnitude estimated. The locations are estimated using a standard locator library and the magnitudes are evaluated using a standard magnitude library.

IDENTIFICATION

GA's components (five programs and one library) are identified as follows:

- *GAassoc*
- *GAconflict*
- *GA_DBI*
- *GAcons*
- *GAgrid*
- *libGA*

STATUS OF DEVELOPMENT

GA, first used operationally in March 1996, is an element of the Prototype International Data Centre (PIDC) at the Center for Monitoring Research (CMR) in Arlington, Virginia, U.S.A. and at the International Data Centre of the Comprehensive Nuclear-Test-Ban Treaty Organization (CTBTO IDC) in Vienna, Austria.

The first version processed only seismic data and ran in the first automatic processing pipeline along with its predecessor Expert System for Association and Location (*ESAL*) [Bac93]. This processing configuration was used during the first year of the Group of Scientific Experts Third Technical Test (GSETT-3). Several upgrades of the subsystem have been made [LeB96]; the first to handle auxiliary seismic data in the

later automatic pipelines and then to adapt for hydroacoustic [LeB97] and infrasonic data types [Kat98]. The subsystem architecture, however, has remained constant as GA has been extended to new data types.

BACKGROUND AND HISTORY

Ronan Le Bras, Walter Nagy, and Jerry Guern of Science Applications International Corporation (SAIC) developed and upgraded GA in the period from 1994 to 2000.

GA was first used operationally in March 1996 as an element of the PIDC processing system at the CMR. The IDC of the CTBTO in Vienna, Austria, first installed the subsystem in May 1998. This document outlines the structure of the GA subsystem delivered with Release 3 to the IDC.

OPERATING ENVIRONMENT

The following paragraphs describe the hardware and commercial-off-the-shelf (COTS) software required to operate GA.

Hardware

GA was designed to run on a Sun UltraSPARC workstation such as the Sun Ultra 5. Typically, the hardware is configured with 128 MB of memory and a minimum of 1 GB of magnetic disk

Commercial-Off-The-Shelf Software

The version of GA described by this document is designed for Solaris 2.7 and ORACLE 8.1.5. The software was designed to be compliant with the database schema described in [IDC5.1.1Rev2].

Chapter 2: Architectural Design

This chapter describes the architectural design of GA and includes the following topics:

- Conceptual Design
- Design Decisions
- Functional Description
- Interface Design

Chapter 2: Architectural Design

CONCEPTUAL DESIGN

GA automatically interprets seismic, hydroacoustic, and infrasonic detection data produced by Station Processing to associate signals from a network of stations and locate seismo-acoustic events using a method similar to generalized beamforming described in [Rin89], [Tay92], and [Leo93]. First, a preliminary event bulletin is produced. This bulletin is subsequently reviewed by human analysts to produce the Reviewed Event Bulletin (REB).

GA was designed to handle the large volumes of data that are needed to monitor compliance with a CTBT. The key feature that permits handling of these large data volumes is a grid-based method used to search for event hypotheses. The grid-based algorithm is a natural way to parallelize the problem; each grid point or sub-section of the earth is treated individually before assembling the different parts.

One of the main considerations for the design of GA is handling the large number of preliminary event hypotheses within a reasonable amount of time. A large set of preliminary event hypotheses is generated in the early stages of *GAassoc*, the functional unit that assembles event hypotheses. To increase efficiency and allow rapid access, propagation knowledge is precomputed and stored within a grid file generated by *GAcons*.

The GA software processes seismic, hydroacoustic, and infrasonic data together and uses only parametric data derived from signal and station processing.

Table 1 lists the automatic bulletin products that are currently supported by GA.

TABLE 1: STANDARD PRODUCTS AVAILABLE THROUGH GA

Product	Availability Time
SEL1	approximately two hours after real time
SEL2	approximately six hours after real time
SEL3	approximately twelve hours after event time

Figure 3 shows the data flow among the five main programs of GA: *GAcons*, *GAgrid*, *GA_DBI*, *GAassoc*, and *GAconflict*.

The *GAcons* program generates two grid files that are used by *GAassoc* and *GAconflict*. The program is a utility that is used to prepare for automatic processing and is not part of the automatic processing operations. *GAcons* must be run to update the grid files every time that a significant change is made to the network, for example, when a new station is added. The first grid file, or Propagation Knowledge Base grid file, contains the dynamic travel-time information necessary to automatically associate detections. The second file, or Static grid file, contains geographical and static data that are used to check deep events against the historical deep seismicity background.

The *GAgrid* program is used to display the content of the Propagation Knowledge Base grid file generated by *GAcons*. The grid file contains both the static grid information and the dynamic propagation information. *GAgrid* is a utility program with a graphical user interface (GUI) component that can be used to review the content of a grid file.

The suite of GA programs, *GA_DBI, GAassoc,* and *GAconflict*, is used in the dynamic pipeline to produce the automatic Standard Event Lists (SEL1, SEL2, and SEL3). These programs use the **arrival**, **amplitude**, **apma,** and accessorily, the **hydro_assoc** and **hydro_arr_group** tables as input and produce the **origin**, **origerr**, **assoc**, **netmag**, **stamag**, and **event** table entries that constitute a bulletin.

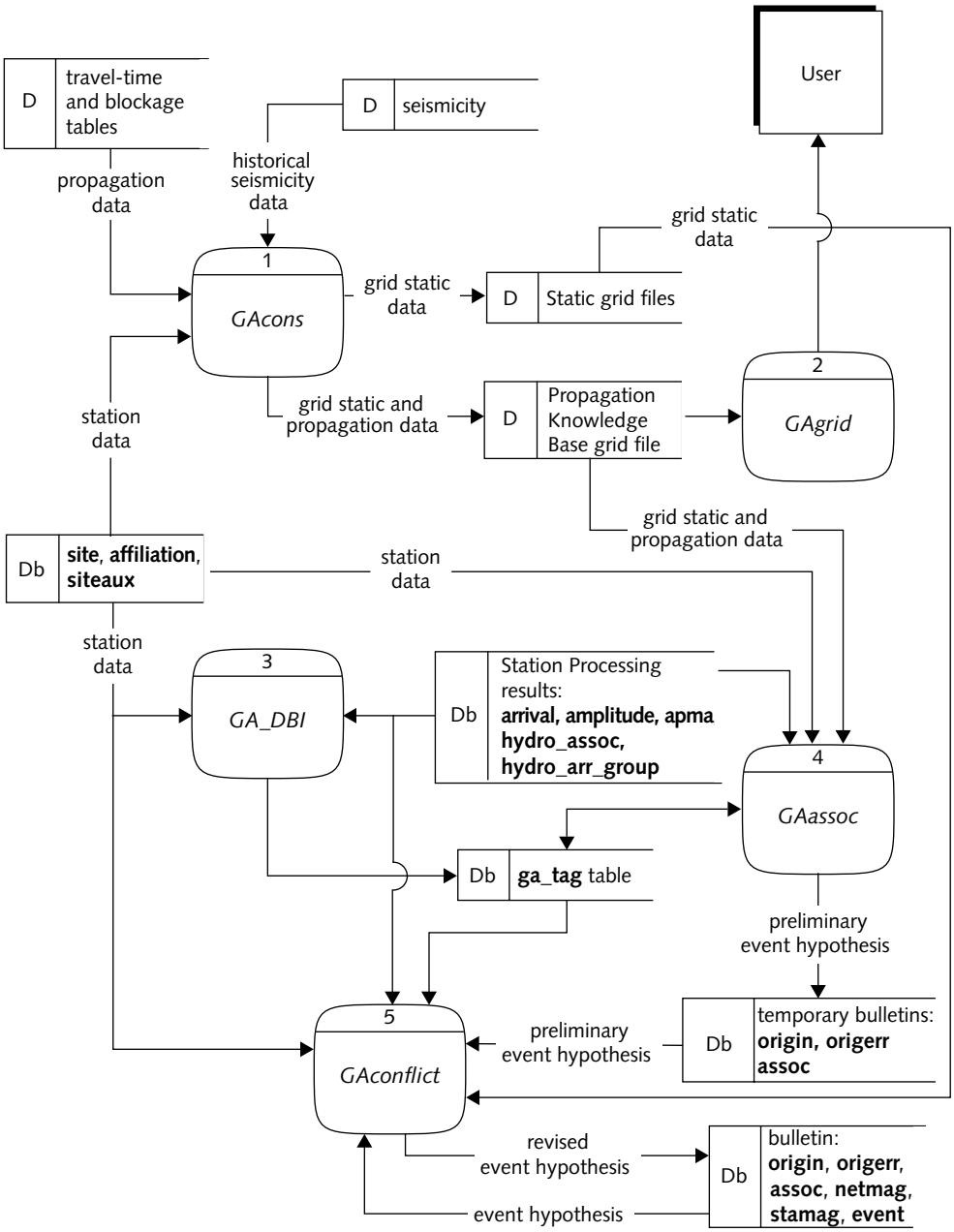


FIGURE 3. HIGH-LEVEL PROCESSING FLOW SHOWING INTERACTIONS AMONG DIFFERENT GA PROCESSES

GAassoc constructs the initial event hypotheses by associating arrivals from different stations using a grid search algorithm. The event hypotheses are placed in a set of temporary tables including **origin**, **origerr**, and **assoc**, which are later read by *GAconflict*. Multiple instances of *GAassoc* can be run in parallel–each instance forming event hypotheses for a different sector of the earth. This parallel design improves processing speed; however, in the configuration used at the IDC with the current IMS network, there is no need to divide the globe into subsectors. Running *GAassoc* using a single global grid achieves the IDC's timeliness goals.

The preliminary bulletin produced by *GAassoc* does not contain conflicting association sets (that is, phases that are associated with more than one event hypothesis) as it performs its own conflict resolution within one sector, but it may contain conflicting associations between sectors and with previously processed time intervals. The *GAconflict* program resolves conflicts between sectors and between time intervals. In addition, the program predicts and associates defining and nondefining phases after relocating initial event hypotheses, and it applies a number of geophysical checks on the associations and events. It modifies or removes associations that do not pass these checks.

GA_DBI performs a few auxiliary functions and is specific to a particular configuration of GA. For example, in the IDC configuration where seismic auxiliary stations are used to improve the location of events determined by the primary seismic network, the *GA_DBI* program is used to tag arrivals from the auxiliary stations in the **ga_tag** table. *GAassoc* and *GAconflict* use the arrival tags to recognize arrivals from the auxiliary stations. These arrivals are not given any weight in the weighted-count calculation. More details are given in "Chapter 7: Detailed Design of GAgrid" on page 95.

DESIGN DECISIONS

The following design decisions pertain to GA.

Programming Language

Each software unit of GA is written in the C programming language unless otherwise noted in this document. The common programming language supports efficient processing and integration with other components of the IDC system.

Global Libraries

The software of GA is linked to the following shared developmental libraries: *libgdi.a*, *libGA.a*, *meschach.a*, *libpar.a*, *libmagnitude.a*, *libprob.a*, *libloc.a*, *libLP.a*, *libinterp.a*, *libgeog.a*, *libaesir.a*, and *libstdtime.a*.

Database

GA uses an ORACLE database to communicate with other processes by obtaining input data and writing results. Input is obtained from the **arrival**, **amplitude**, **apma**, **hydro_assoc**, and **hydro_arr_group** database tables in the station processing account of the database as well as static information from the **site**, **affiliation**, and **siteaux** tables. The output is written to the **origin**, **origerr**, **assoc**, **event**, **netmag**, and **stamag** tables in the event list accounts (SEL1, SEL2, and SEL3) of the database. One instance of the output tables exists for each of the event list accounts. *GAconflict* also reads tables **origin**, **origerr**, and **assoc** from the previous event list account (from SEL1 for the SEL2 processing).

Interprocess Communication (IPC)

GA does not use the IPC system internally; however, the *GA_DBI*, *GAassoc*, and *GAconflict* programs are executed in the context of individual pipelines and are controlled by the IPC-based Distributed Application Control System (DACS).

Filesystem

GA's use of the filesystem includes reading the two grid files generated by *GAcons*, the attenuation files for computation of the magnitudes, the `slowamp.P` file for probability of detection computation, the parameter files, and the travel-time table files. GA also writes informational and error messages to log files.

Design Model

The design of GA is primarily influenced by timeliness, flexibility, and reliability requirements. The timeliness requirements are derived from the requirement to produce an automatic bulletin to facilitate the analyst's task of providing the final bulletin. The basic timeliness requirement is that the software be able to process data and publish a bulletin faster than real time on average. The timeliness requirement and the anticipation of increased data rates from the growing IMS network led to the partition of GA into two major components: *GAassoc* and *GAconflict*. *GAassoc* can be run as several parallel instances, each processing the grid points on a subsector of the earth, thus reducing the total processing time. Thus far, it has not been necessary to use this feature to process IMS data at the IDC.

The software is sufficiently flexible to allow GA to be configured in a multi-pipeline model where bulletins are refined after each successive pass. It can also be configured to handle single technology processing or simultaneous processing of the three waveform technologies with a parameter-settable level of mixing of the three different technologies. Numerous details of the processing can also be controlled by user parameters.

The software is designed to run on discrete intervals of time. The configuration at the IDC uses 20-minute intervals in each of three processing pipelines.

Reliability is an important requirement for a software system that must process the large quantity of data from the IMS network and is called nine times every hour on a continuous basis. The subsystem was designed to handle a variety of potential failure modes.

An important design decision for GA was to modularize and group the main functions of GA within a single library. Processes within the main functional units are shared among these different functional units, and they are grouped in the *libGA* library. For instance, the process that locates all preliminary event hypotheses within *GAassoc* and *GAconflict* uses the same function from the *libGA* library for both programs. Other examples are the functions that read all GA control parameters and resolve association conflicts.

Database Schema Overview

GA uses the ORACLE database for the following purposes:

- to access the input database tables (**arrival**, **amplitude**, **apma**, **hydro_assoc**, and **hydro_arr_group**) to obtain the results of Station Processing
- to access the static database tables (**site**, **siteaux**, and **affiliation**) to get the station and network information
- to access the output database tables (**origin**, **origerr**, **assoc**, **event**, **netmag**, and **stamag**) to write bulletin information
- to exchange data among *GA_DBI*, *GAassoc*, and *GAconflict* (**ga_tag** and temporary tables **origin_temp_ga**, **origerr_temp_ga**, and **assoc_temp_ga**)

Table 2 shows the tables used by GA. The Name field identifies the database table. The Mode field is "R" if GA reads from the table and "W" if the subsystem writes to the table.

TABLE 2: DATABASE TABLES USED BY GA

Name	Mode	Description
arrival	R	contains summary information about arrivals
amplitude	R	contains arrival-based and origin-based amplitude measurements
apma	R	contains results of particle motion analysis for a specific detection

TABLE 2: DATABASE TABLES USED BY GA (CONTINUED)

Name	Mode	Description
hydro_assoc	R	contains information that connects arrivals to a hydroacoustic group of arrivals
hydro_arr_group	R	contains information about hydroacoustic groups of arrivals
site	R	contains station location information; names and describes a point on the earth where measurements are made (for example, the location of an instrument or array of instruments); contains information that normally changes infrequently, such as station location
siteaux	R	contains additional site-dependent parameters that are not included in the **site** table
affiliation	R	groups stations into networks
origin	R/W	contains information describing a derived or reported origin for a particular event; GA writes an **origin** record for each new or relocated event hypothesis
origerr	R/W	contains summaries of the confidence bounds for origin estimates; GA writes a record for each event hypothesis
assoc	R/W	contains information that connects arrivals (entries in the **arrival** table) to a particular origin (an entry in the **origin** table); GA writes **assoc** records for every arrival associated with an event hypothesis
event	R/W	contains a list of events; GA writes an **event** record for each event hypothesis
netmag	R/W	contains estimates of network magnitudes of different types for an event; each network magnitude has a unique *magid*; station magnitudes used to compute the network magnitude are in the **stamag** table; for each event hypothesis GA writes a **netmag** record for each magnitude type that it estimates
stamag	R/W	contains station magnitude estimates based upon measurements made on specific seismic phases; values in **stamag** are used to calculate network magnitudes stored in **netmag**; for each association of an event hypothesis GA writes one or more **stamag** records in the **stamag** table

TABLE 2: DATABASE TABLES USED BY GA (CONTINUED)

Name	Mode	Description
ga_tag	R/W	contains information on the use of arrivals and origins in GA
origin_temp_ga	R/W	used by GA to store temporary origins to be communicated between *GAassoc* and *GAconflict*
origerr_temp_ga	R/W	used by GA to store temporary origin error information for the origins in **origin_temp_ga**
assoc_temp_ga	R/W	used by GA to store temporary associations for the origins in **origin_temp_ga**

Database Entity-relationship Diagram

Figure 4 is an entity-relationship diagram that shows all the database tables used by GA.

FUNCTIONAL DESCRIPTION

Figure 5 shows the main functional units of GA and the interactions among them, with the database tables and the grid files. The *GAcons* process is a stand-alone program that builds the Propagation Knowledge Base grid file and the Static grid file to be used by the pipeline-activated programs *GAassoc* and *GAconflict*. The *GAgrid* program is a GUI used to visualize the GA grid file containing the knowledge base built by *GAcons*. *GA_DBI* is a mission-specific program customized for the IDC configuration. In that configuration, the main function of the program is to tag arrivals from the auxiliary network so that they are recognized as such.

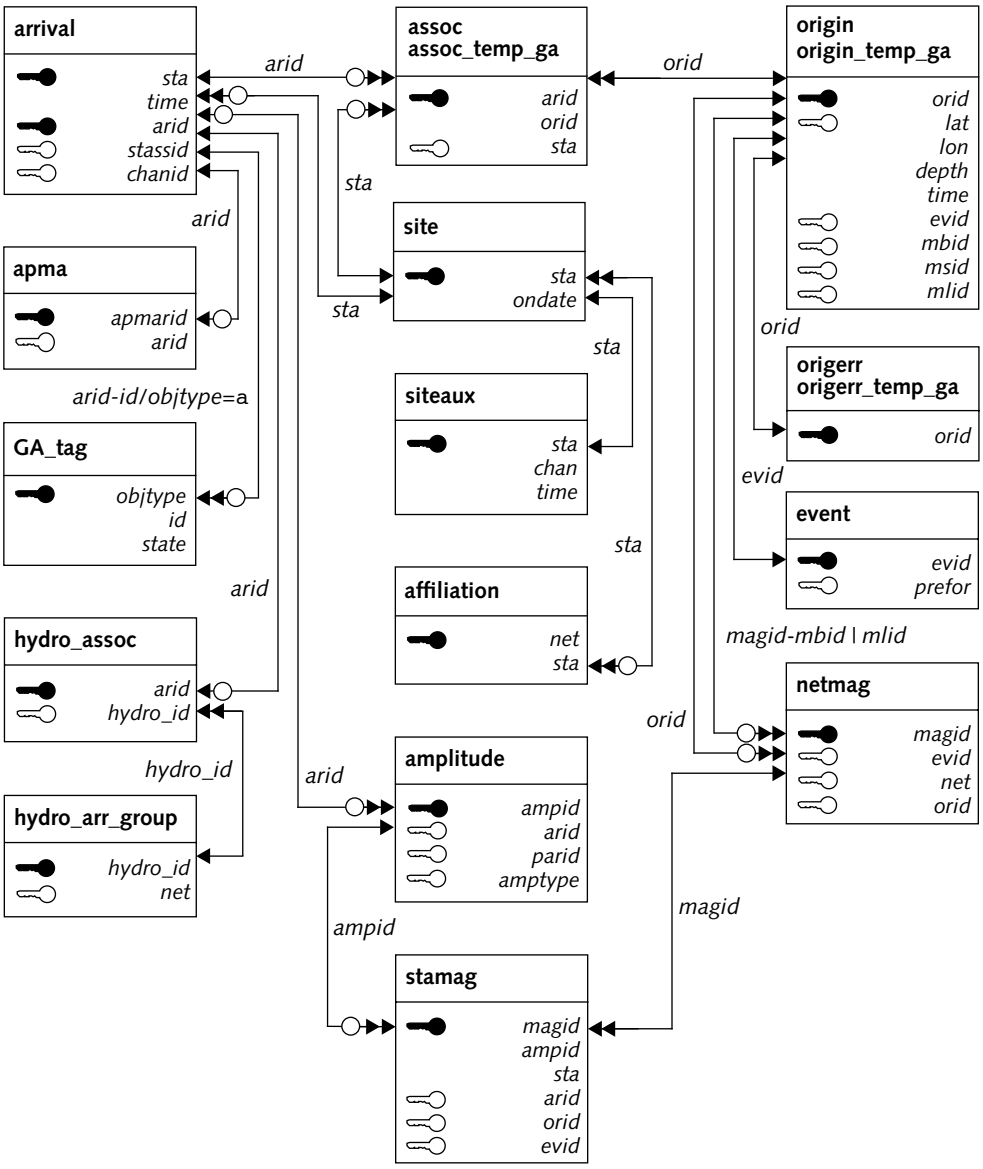


FIGURE 4. GA DATABASE TABLE RELATIONSHIPS

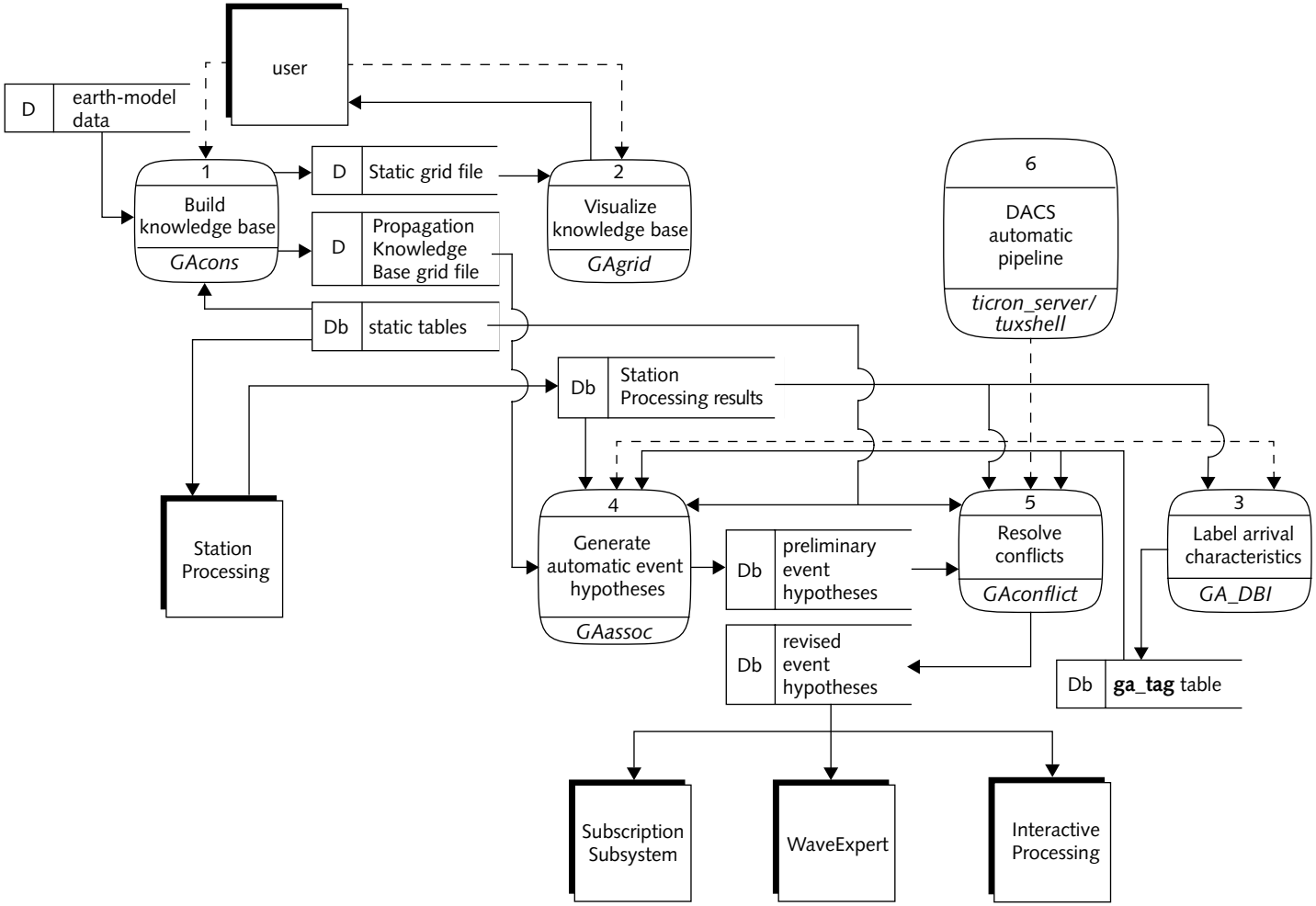


FIGURE 5. GA FUNCTIONAL DESIGN

Building the Knowledge Base

GAcons is a stand-alone program used to generate two separate files used by GA during pipeline processing. The first file, the Propagation Knowledge Base grid file, contains pre-computed information pertaining to the propagation of the various types of phases among potential source regions and stations. That file is used only by *GAassoc* and is essential to build the initial association sets. The second file, the Static grid file, contains geographical information for the grid used in the association process. This file is used only by *GAconflict* to check on the location of deep events in areas of deep seismicity. The input to *GAcons* is the travel-time information and network information along with a set of control parameters to produce the two grid files.

Visualizing the Knowledge Base

GAgrid provides a GUI for examining the contents of the grid file, which contains the knowledge base. The graphical interface displays grid cell boundaries, continental and political boundaries, and stations of the network on a global map. For each cell, a list of stations, phases, and information pertaining to the paths between stations and cell are displayed in an alphanumerical form. *GAgrid* can display the travel-time information for each cell and phase contained in the Propagation Knowledge Base grid file.

Generating New Automatic Events

GAassoc generates new automatic event hypotheses from the results of Station Processing. The events are generated via an exhaustive grid-search algorithm. In an initial phase, all possible events satisfying a set of criteria for the set of input arrivals are formed by examining each grid cell in the input file, one at a time. The initial set of events is then examined and pruned to eliminate redundancies. The next step is location and outlier analysis, which removes misfits. Finally, conflict resolution is applied to the set of located events to arrive at the final set of self-consistent, newly created event hypotheses. The results are written to temporary database tables that are refined by *GAconflict*.

Resolving Conflicts

GAconflict resolves conflicts between different sets of GA event hypotheses and refines the results. *GAconflict* reads up to three sets of bulletin tables (**origin**, **origerr**, and **assoc**) as input. The first set of bulletin tables are the temporary tables written by *GAassoc*; the second set of tables are the output bulletin tables written by *GAconflict* for the prior processing interval, and the third (optional) set of tables are the bulletin tables from a previous pipeline, which can be used as input to the current pipeline. (At the IDC, SEL1 tables are read when producing the SEL2, and SEL2 tables are read when producing the SEL3.) In addition to the bulletin tables, *GAconflict* also reads the **ga_tag**, **arrival**, **amplitude**, **apma**, **hydro_assoc**, and **hydro_arr_group** database tables. The main processes performed by *GAconflict* are conflict resolution between adjacent time intervals, prediction of defining and non-defining phases, and seismological consistency checks on the automatic event hypotheses. The output tables are **origin**, **origerr**, **assoc**, **event**, **netmag**, and **stamag**, which collectively contain the automatic bulletin information produced by GA.

Setting Up Arrival Tags

GA_DBI performs mission-specific tagging of arrivals for use by the other GA processes. The input to *GA_DBI* are the **arrival** and **affiliation** database tables. The output table is **ga_tag**. At the IDC, the main function of *GA_DBI* is to tag arrivals from the auxiliary seismic network so that they are recognized as such by *GAassoc* and *GAconflict*.

INTERFACE DESIGN

This section describes GA's interfaces with other IDC systems, external users, and operators.

Interface with Other IDC Systems

The main mechanism for exchanging data between GA and other IDC subsystems is through the database and in particular the database tables shown in Figure 2 on page 5. GA also reads data from the filesystem in a standard manner consistent

with the parameter interface and travel-time handling system. GA is invoked on a specific schedule by the DACS. Typically, GA is called every 20 minutes in each pipeline.

Interface with External Users

In its standard IDC configuration, GA is run in an automatic pipeline to produce automatic bulletins and has no explicit user interface. However, external users can use GA on a standalone basis to conduct special studies. In this case, GA is invoked from the command line with appropriate parameters.

Interface with Operators

GA is normally run in the context of the DACS-controlled automatic pipeline and is one of the processes in the SEL1, SEL2, and SEL3 pipelines, along with *WaveExpert* and *DFX-originbeam*. Operator intervention is minimal in a normal context and is only necessary in case of processing failure on a particular time interval or to stop and restart a pipeline. Processing in the three automatic pipelines is monitored through the *WorkFlow* GUI program, which is the main monitoring tool at the disposal of pipeline engineers. The interface between GA and the *WorkFlow* monitoring tool is the **interval** table, which contains the following states relating to GA processing:

`GA_DBI-done`, `GA_DBI-started`, `GAassoc-done`, `GAassoc-started`, `GAconf-done`, `GAconf-started`, `GA-failed`, and `GA-retry`.

Tuxpad is a GUI application that can be used to start and stop the automatic pipelines. Instructions on how to operate *Tuxpad* are given in [IDC6.2.1].

The log files written by the GA applications are also used for monitoring and diagnostics. The log files contain error message output, diagnostic output, and information on the timing of different processes within each GA application, including explicit timing of database reads.

Chapter 3: Detailed Design of GAassoc

This chapter describes the detailed design of *GAassoc* and includes the following topics:

- Data Flow Model
- Processing Units
- Database Description

Chapter 3: Detailed Design of GAassoc

DATA FLOW MODEL

GAassoc uses arrival data for the current time interval and the grid information produced by *GAcons* to generate self-consistent sets of associated arrivals that may have resulted from a single event. Association sets that pass various acceptance tests and the process of conflict resolution become preliminary event hypotheses. These preliminary events are written to an intermediate set of temporary tables (**origin, origerr**, and **assoc**) called the *GA_tables*. Several instances of *GAassoc* may be run in parallel, each on a different sector, or region, of the earth. Each instance writes its results to the same intermediate set of temporary tables.

Figure 6 shows the data flow of *GAassoc* including the main processing units as well as the data stores used as input and output by these processing units. The internal `Driver` structure data store serves as a basic and standard data representation that is used from the initiation of the association sets to the final output as a temporary bulletin. A string of processes from the *Eliminate Redundant Events* to the *Write Event Hypotheses to Database* use this basic data structure as input and output, which allows for a modular design. This design also allows re-use of the processes within other programs, when appropriate, with the desirable effect of cutting down on maintenance costs. Some of the same processes, for example, *Resolve Conflicts*, are also used by *GAconflict* (see "Chapter 6: Detailed Design of GAcons" on page 81). All of the processes shown in Figure 6 take the list of GA parameters as input. The GA parameters are read by a single process, which is shared with *GAconflict*.

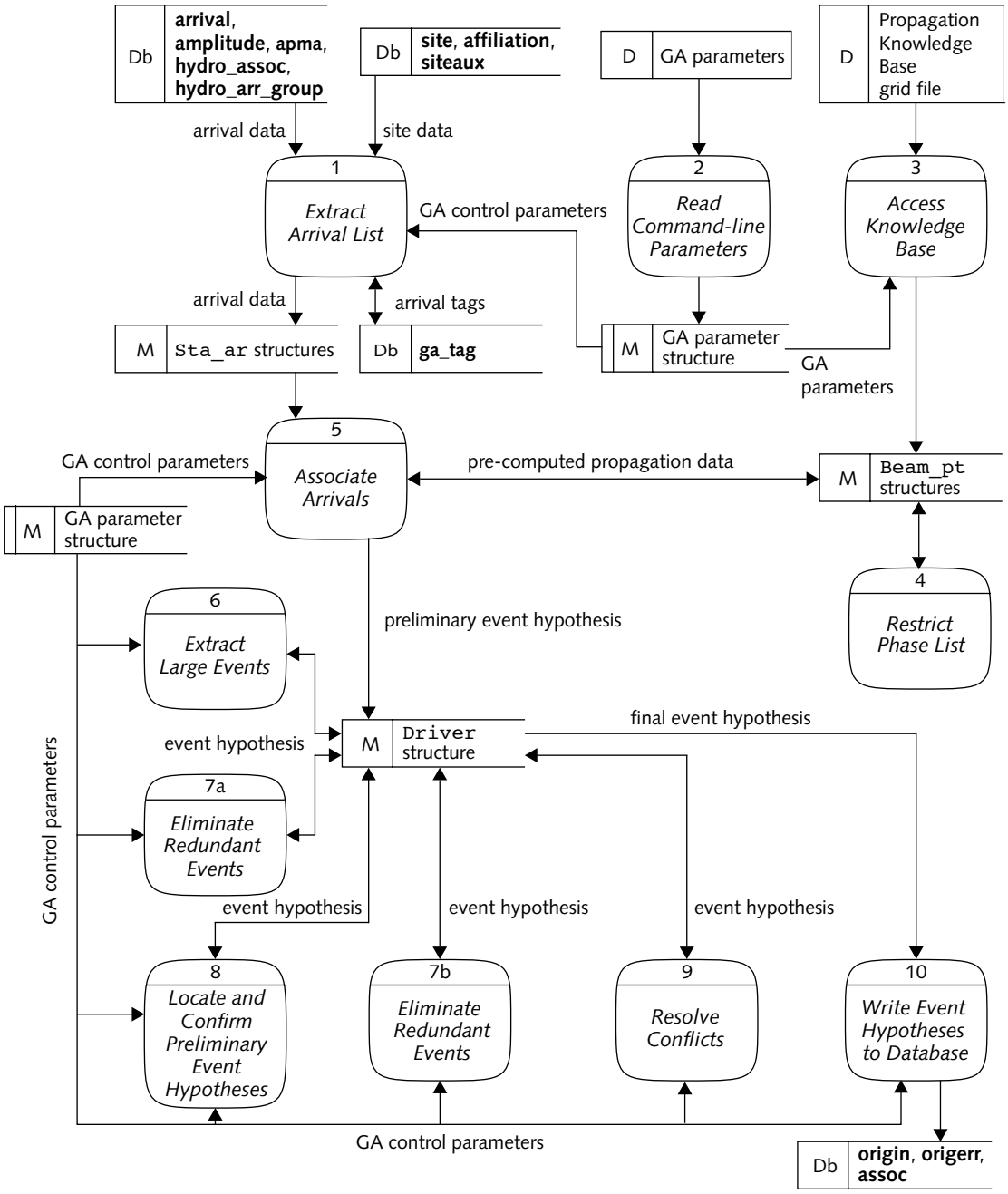


FIGURE 6. GAASSOC DATA FLOW

Table 3 shows the sequence of processes that leads to the production of a newly formed event in the temporary bulletin database populated by *GAassoc*.

TABLE 3: PROCESSING UNITS AND CORRESPONDING C FUNCTIONS

Processing Unit	Process Number on Data Flow	Function Name

Extract Arrival List	1	`GA_assoc()`
		`GA_build_arrival_query()`
		`GAarrivals()`

Read Command-line Parameters	2	`GA_read_par()`
Access Knowledge Base	3	`GA_file()`
Restrict Phase List	4	`GA_restrict_phases()`
Associate Arrivals	5	`GA_assoc_loop()`
Extract Large Events	6	`GA_extirp_large()`
Eliminate Redundant Events	7a, 7b	`GA_redundancy_check()`
Locate and Confirm Preliminary Event Hypotheses	8	`GA_locate_list()` `GA_locate()`
Resolve Conflicts	9	`GA_cluster()` `GA_assoc_based_CR()`
Write Event Hypotheses to Database	10	`GA_assoc()`

PROCESSING UNITS

GAassoc consists of the following processes, as shown in Figure 6:

- *Extract Arrival List* (1)

  This process reads station and arrival data from the database to produce `Sta_ar` structures, an internal representation of the arrivals for the time interval being processed.

- *Read Command-line Parameters* (2)

  This process reads parameters from the command line and from par files and stores the values internally.

- *Access Knowledge Base* (3)

  This process reads from the Propagation Knowledge Base grid file and produces the internal representation of this information in the `Beam_pt` structures.

- *Restrict Phase List* (4)

  This process restricts the phase list in the `Beam_pt` structures to conform to the list specified in the set of *GAassoc* parameters.

- *Associate Arrivals* (5)

  This process forms the initial set of associated arrivals. Associating arrivals is a fundamental step [IDC5.2.1]. It uses the `Sta_ar` structures and the `Beam_pt` structures as input and produces the initial set of preliminary events represented in a linked list of `Driver` structures.

- *Extract Large Events* (6)

  This process identifies, locates, and extracts event hypotheses with a large number of defining detections. The purpose of this process is to reduce processing time when there are large events in the time interval.

- *Eliminate Redundant Events* (7a, 7b)

  This process eliminates redundant event hypotheses from a set of event hypotheses. This process appears twice in Figure 6 on page 27; once before and once after the *Locate and Confirm Preliminary Event Hypotheses* process.

- *Locate and Confirm Preliminary Event Hypotheses* (8)

  This process estimates an event's origin time, latitude, longitude, and depth based on the travel-time and slowness attributes of the detections in its association set. Confirmation is the process that verifies that the events meet the required criteria including a weighted count of arrival attributes, an arrival-quality test, and a probability-of-detection test. The weighted-count test compares a weighted sum of arrival attributes

against a threshold value. The arrival-quality test compares the sum of the arrival-quality functions over all defining arrivals in the event hypothesis with a threshold value. The arrival-quality function is a rational function of the uncertainty in the measure of slowness and the distance between the event and the station. It is empirical in nature and has been found to help in reducing the number of false event hypotheses constructed from stochastically consistent sets of arrivals.

- *Resolve Conflicts* (9)

  This process eliminates the ambiguity and inconsistency of associating an arrival to multiple events. After this process, an arrival is associated to no more than one event.

- *Write Event Hypotheses to Database* (10)

  This process writes the tables composing the temporary bulletin.

The following paragraphs describe the design of these processes.

Extract Arrival List

The purpose of this process is to build and execute the queries that extract the correct list of arrivals from the database and insert them into internal data structures for further processing. The process reads the arrivals produced by Station Processing from the database using standard system libraries (*libgdi*).

The network names and the number of networks for each technology are flexible, and the process that builds the arrival query is re-usable in other programs.

Input/Processing/Output

The inputs to this process include control parameters read from a parameter file, static database tables (**affiliation**, **site**, and **siteaux**) and dynamic database tables (**arrival, amplitude, apma, hydro_assoc**, and **hydro_arr_group**), as indicated in Figure 6 on page 27. The processing populates the `Sta_ar` data structures and updates the database **ga_tag** table, which constitute the output of this process.

Control

The *Extract Arrival List* process is started when *GAassoc* is started. When severe error conditions are encountered, for instance with database input, the program writes an informative message to standard output, which is usually directed to the log file, then exits. This process consists of several modules and calls to database interface functions. Control stays with the main *GAassoc* program after successful completion of the process.

Interfaces

The *Extract Arrival List* process interfaces with the *libgdi* library to acquire its database input. All of the database interface functions from *libgdi* are called in the main `GA_assoc()` program. `GA_build_arrival_query()` function builds a query string based on the input control parameter for the current interval, and the string is passed to the *libgdi* library for execution. Finally, the results of the query are parsed by `GAarrivals()`, which also populates the `Sta_ar` internal structures.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process.

Read Command-line Parameters

This process interfaces with the *libpar* library and populates the internal parameter structures. GA uses a large number of parameters to configure its processing, and some of the parameters are used by several of the programs within the subsystem. The parameter input is modularized in such a way that it can be used by several programs. Currently, *GAassoc* and *GAconflict* use this process. The purpose of the *Read Command-line Parameters* process is to parse the command line, extract the parameters, and populate the internal parameter structure.

Input/Processing/Output

The inputs to this process are control parameters read from a parameter file or directly from the command line. The processing parses these parameters using the *libpar* library and writes an internal data structure containing the GA parameters.

Control

The *Read Command-line Parameters* process is started when *GAassoc* is started. When severe error conditions are encountered, for example with database input, the program writes an informative message to standard output, which is usually directed to the log file, then exits. If successful, control is returned to the main *GAassoc* program.

Interfaces

The *Read Command-line Parameters* process interfaces with the *libpar* library to parse parameters from the command line or parameter file(s).

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. In this particular case, the errors that may result from calling the subroutines of the *libpar* library are flagged and captured, an error message is issued indicating the parameter that the process failed to read properly, and the process exits with a failure condition.

Access Knowledge Base

The purpose of this process is to parse the Propagation Knowledge Base grid file to read the information relevant to the cell being processed.

Input/Processing/Output

The inputs to this process are the Propagation Knowledge Base grid file written by the *GAcons* program and the GA control parameters. The process reads the information from the binary grid file and writes internal `Beam_pt` structures.

Control

The *Access Knowledge Base* process is modularized and called within the *Associate Arrivals* process. Upon exit control is returned to that process. When severe error conditions are encountered the program writes an informative message to standard output, which is usually directed to the log file, then exits.

Interfaces

The *Access Knowledge Base* process uses standard C I/O libraries to access the data in the Propagation Knowledge Base grid file.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. In this particular case, the errors that may result from the memory allocation functions called from within the process are captured and passed to the calling routine, which issues an informative message including an error code and exits.

An error code of `20`, `21`, or `22` indicates an I/O error from reading the grid file. A value of `20` is given when an attempt at opening the file fails. Values of `21` and `22` are given when an attempt at reading the file fails. If any of these error conditions occur, the user can check that the file exists at the location specified by the control parameters and that permissions are sufficient to access it. If the access settings are confirmed and the error still occurs, the file might have been corrupted (for instance during a file transfer) and may need to be re-generated by *GAcons*.

Error codes other than the ones mentioned above (in other words, from `23` through `30`) are the result of a memory allocation problem.

Restrict Phase List

The Propagation Knowledge Base grid file contains information about a number of phases. More phases are in the grid file than are necessary for the purpose of forming event hypotheses. The purpose of the *Restrict Phase List* process is to limit the list to the phases used by *GAassoc*.

Input/Processing/Output

The input to this process is the `Beam_pt` data structure, which contains the propagation knowledge for a given cell point as written by the *GAcons* program. The processing consists of pruning the list read from the Propagation Knowledge Base grid file and restricting it to the list given as a control parameter to *GAassoc*. The output of the process is a modified `Beam_pt` data structure.

Control

The *Restrict Phase List* process is modularized and called within the *Associate Arrivals* process. Upon exit control is returned to that process. When severe error conditions are encountered the program writes an informative message to standard output, which is usually directed to the log file, then exits.

Interfaces

The *Restrict Phase List* process does not interface with any external systems.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process.

Associate Arrivals

The *Associate Arrivals* process is the algorithmic heart of *GAassoc*. This process forms preliminary event hypotheses using an exhaustive grid search algorithm. The following are system requirements for this process:

- Form all possible self-consistent, independent event hypotheses that satisfy the criteria of acceptance.
- Be robust enough to run for all time intervals.
- Provide a high-quality bulletin to assist the work of analysts.
- Form events in a timely manner.

Input/Processing/Output

The inputs to the *Associate Arrivals* process are the `Beam_pt` structures and the `Sta_ar` structures, as shown in Figure 6 on page 27. The `Beam_pt` structures contain the information read from the grid file pertaining to the propagation of seismic and acoustic phases between one grid cell and all the stations in the networks. The `Sta_ar` structures contain the information that characterizes the arrivals read from the database.

The following paragraphs briefly describe the algorithm used to form preliminary events. A more complete and detailed description is available in [IDC5.2.1].

The algorithm works one grid cell at the time. Event hypotheses are formed for a cell by identifying a driver arrival from one of the stations in the list of stations that may see the first-arrival for an event located in the grid cell. This list of stations, which are allowed to have the first detected arrival (driver), consists of the first *N* stations ordered by distance from the grid cell; *N* is a settable parameter. A driver arrival must have a slowness vector that is compatible with the current cell. After a driver arrival has been identified, corroborating arrivals are associated with the event hypothesis, where the corroborating arrivals are from stations other than the driver arrival station. Compatibility of the time and slowness attributes with the preliminary event hypothesis formed by the driver arrival are first established. Compatibility of the event formed by the driver and each corroborating arrival with

the location of the grid cell are then tested. All stations present in the grid file are systematically searched for corroborating arrivals after the driver arrival has been established.

The output of the *Associate Arrivals* process is a linked list of `Driver` structures. The `Driver` structures (see Table 5 on page 39) contain the preliminary event hypotheses formed by the process. All other processes that follow within *GAassoc* both read and write this type of data structure. The structure is also used throughout *GAconflict*. In the course of being transformed by the successive processes, the `Driver` structures evolve and may acquire attributes such as an **origin** record and **assoc** records after passing through the *Locate and Confirm Preliminary Event Hypotheses* process.

Control

The *Associate Arrivals* process is one of an integrated suite of processes that are triggered sequentially when *GAassoc* is started. The process is modularized in subroutine `GA_assoc_loop()`; it starts when the subroutine is invoked, and it finishes either when the subroutine returns control to the main *GAassoc* program or exits with an error condition.

Interfaces

The primary purpose of the *Associate Arrivals* process is to form the initial list of preliminary event hypotheses using grid and arrival information in the `Beam_pt` (Table 16 on page 90) and `Arrival_Inf` (Table 8 on page 42) structures. Figure 7 on page 38 shows the relationships between some of the most important data structures within the *GAassoc* program, in particular the relationships between the `Driver` (Table 5 on page 39), the `Cor_sta` (Table 6 on page 41), the `Sta_ar` (Table 7 on page 42), and the `Arrival_Inf` (Table 8 on page 42) data structures. The information that is read at the beginning of the *GAassoc* session and remains static throughout the session is indicated by the gray area. The squares with a dot represent pointers. The arrows show the link between the pointer and the object to which they point.

Associate Arrivals is the process that first populates the data structures shown outside of the gray area in Figure 7 on page 38. The `Driver` structures are first constructed by identifying a driver arrival and then establishing a corroborating list of arrivals. The `Driver` structure keeps track of the pointer to the driver arrival as well as the pointer to the `Sta_Ar` structure for the driver-arrival station. The corroborating arrivals are organized in a linked list of `Cor_Sta` structures where the pointer to the leading member of the list is part of the `Driver` structure. The `Cor_Sta` structure itself contains pointers to the `Arrival_Inf` structure for the corroborating arrival and to the `Sta_Ar` structure for the station of the corroborating arrival.

Table 4 summarizes the primary content of the data structures used by *GAassoc* and *GAconflict*. Tables 5 through 8 provide the detailed content of these C structures.

TABLE 4: PRIMARY DATA CONTENT IN STRUCTURES USED BY GA

Data Structure Name	Primary Data Content
`Driver`	This structure contains preliminary event hypothesis data and is organized in an evolving linked list throughout *GAassoc* and *GAconflict*. The linked list is passed from process to process in *GAassoc* and *GAconflict* and can be thought of as an object on which methods (or processes) perform their operations.
`Cor_sta`	This structure keeps track of the corroborating arrivals within a `Driver` structure and is organized in a linked list whose anchor (pointer to the first element) is an element of the `Driver` structure. The structure contains pointers to the corroborating arrivals.
`Arrival_Inf`	This structure stores data that characterize an arrival from database tables **arrival**, **apma**, **amplitude**, and **hydro_assoc**. The structures are populated once and remain unchanged for the duration of *GAassoc* processing.
`Sta_ar`	This structure contains information about the stations with arrivals in the interval to be processed by GA and includes a pointer to the array of `Arrival_Inf` containing the arrivals for that station.

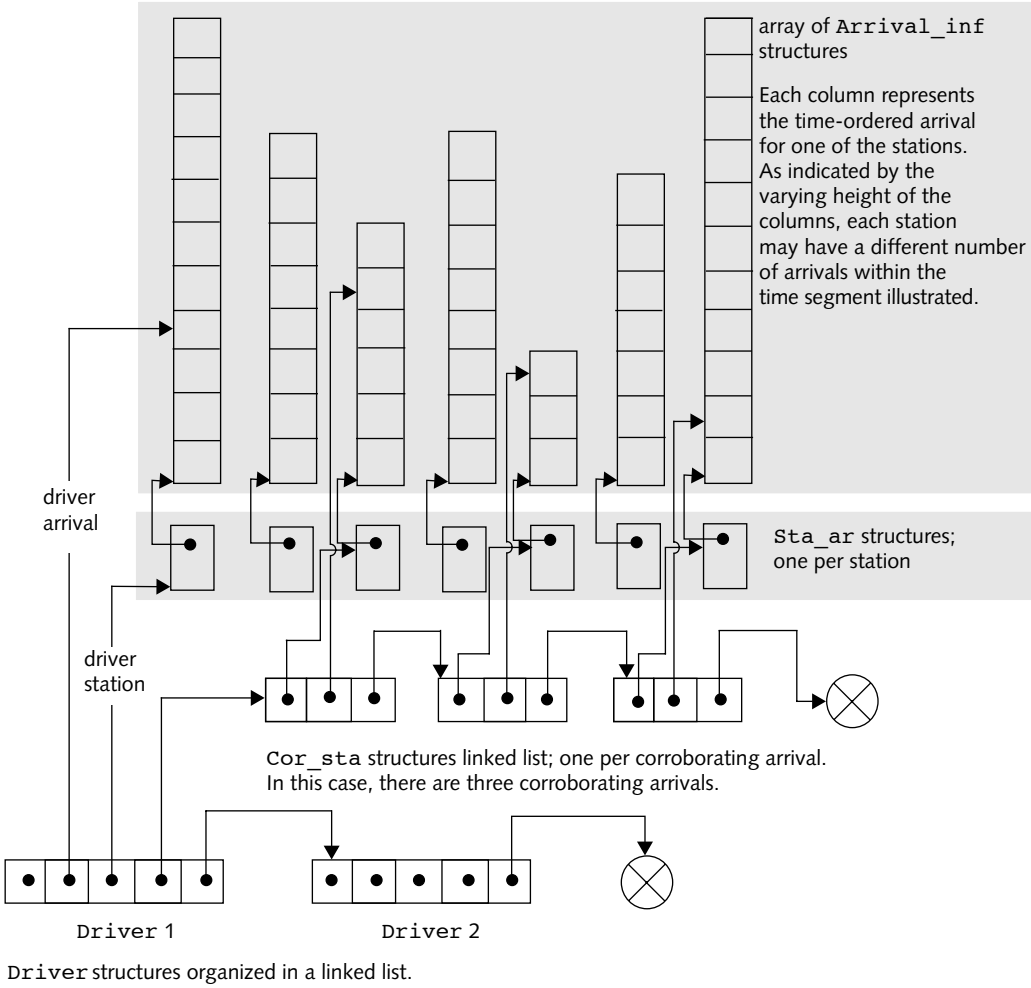


FIGURE 7. RELATIONSHIPS AMONG GAASSOC DATA STRUCTURES

TABLE 5: DRIVER STRUCTURE

Type	Name	Description of Structure and Data Members
`Beam_pt*`	*bp*	pointer to Beam-point
`char`	*ph_id[ ]*	phase identification (ID) for driver
`StaPt*`	*stpt*	pointer to station information for Beam-point
`Phas_Inf*`	*phspt*	pointer to phase information for this driver-phase Beam-point
`Sta_Ar*`	*sta*	pointer to station arrival structure for first-arrival station
`Arrival_Inf*`	*ar*	pointer to `Arrival_Inf` structure for the driver arrival for this `Driver` structure
`Cor_Sta*`	*csta*	pointer to corroborating stations and arrivals list
`Dr_list*`	*drl*	pointer to the `Dr_list` structure (see Table 9 on page 49) for the driver arrival
`double`	*or_time*	origin time for the driver
`double`	*or_tmin*	minimum origin time for the driver
`double`	*or_tmax*	maximum origin time for the driver
`double`	*dr_mag*	driver magnitude
`double`	*qfact*	quality factor; this is the combined probability for all corroborating phases to be associated with this driver
`double`	*ar_qual*	quality of the driver arrival (see "Locate and Confirm Preliminary Event Hypotheses" on page 49)
`double`	*cr_ev_qual*	event-based quality as measured for conflict resolution
`double`	*cr_hydro_qual*	event-based quality for events containing hydroacoustic arrivals
`double`	*cr_infra_qual*	event-based quality for events containing infrasonic arrivals

TABLE 5: DRIVER STRUCTURE (CONTINUED)

Type	Name	Description of Structure and Data Members
`double`	*cr_ev_qual*	event-based quality as measured by conflict resolution "goodness-of-fit"
`double`	*chi_prob*	chi-squared probability number from the locator
`double`	*res_norm*	network-based, probability-of-detection residual norm, measured as residual/σ
`double`	*weight*	current weight of driver obtained by adding all weights for associated arrivals
`double`	*ar_qual_weight*	weight of the preliminary event obtained by adding all arrival-quality weights
`int`	*num_obs*	total number of arrivals for this driver, including the driver
`int`	*num_def*	number of defining detections for this driver, including the driver
`int`	*nsta_mag*	number of stations used to compute m_b
`int`	*nsta_ml*	number of stations used to compute M_L
`int`	*input*	flag identifying the origin of the preliminary event; the event is either newly created in *GAassoc*, inherited from a previous bulletin, or inherited from a previous interval of the current bulletin
`int`	*unique_id*	unique identifying number for the driver (internal to GA)
`double`	*mag_sig*	uncertainty in m_b computation
`double`	*mag_sigml*	uncertainty in M_L computation
`Origin*`	*origin*	pointer to structure containing **origin** record
`Origerr*`	*origerr*	pointer to structure containing **origerr** record
`Assoc*`	*assoc*	pointer to structure containing **assoc** record
`Assoc_CR*`	*assoc_cr*	pointer to association-based conflict resolution information

TABLE 5: DRIVER STRUCTURE (CONTINUED)

Type	Name	Description of Structure and Data Members
`Bool`	*locked_event*	flag indicating whether or not the event has been locked by an analyst
`int`	*modified*	book-keeping field to allow processes to track their own effects on a `Driver` list
`Driver*`	*next*	pointer to next `Driver`
`Driver*`	*previous*	pointer to previous `Driver`

TABLE 6: COR_STA STRUCTURE

Type	Name	Description of Structure and Data Members
`Sta_Ar*`	*sta*	pointer to `Sta_Ar` structure for first-arrival station
`Arrival_Inf*`	*ar*	pointer to `Arrival_Inf` structure for first-arrival station given driver
`double`	*qual*	quality of the chi2 association
`double`	*ar_qual*	quality of the arrival (delslo-distance based)
`double`	*mag*	station magnitude of corroborating association
`char`	*ph_id[ ]*	phase ID for corroborating arrival
`char`	*def[2]*	defining/nondefining flag (d) if defining, else (n)
`Dr_list*`	*drl*	pointer to the `Dr_list` structure for the driver arrival
`Cor_Sta*`	*next*	pointer to next corroborating station

TABLE 7: STA_AR STRUCTURE

Type	Name	Description of Structure and Data Members
`char`	*name*	station name
`int`	*nar*	number of arrivals associated to this station
`int`	*site_index*	index into **site** table
`Bool`	*array*	`TRUE`, if station is an array
`Bool`	*hydro*	`TRUE`, if station is a hydroacoustic station
`Bool`	*infra*	`TRUE`, if station is an infrasonic station
`Arrival_Inf*`	*ar*	pointer to array of arrivals

TABLE 8: ARRIVAL_INF STRUCTURE

Type	Name	Description of Structure and Data Members
`long`	*arid*	*arid* for this arrival
`double`	*time*	epoch time of arrival
`char`	*iphase*	reported initial phase
`char`	*ml_chan*	channel read from **amplitude** table for M_L calculation
`char`	*mb_chan*	channel read from **amplitude** table for m_b calculation
`double`	*deltim*	measured time uncertainty
`double`	*deltim_mod*	modelled time uncertainty
`double`	*azimuth*	measured azimuth
`double`	*delaz*	uncertainty in azimuth
`double`	*slow*	measured slowness
`double`	*delslo*	uncertainty in slowness
`double`	*amp*	measured amplitude from the **amplitude** table (*amptype* = `A5/2`)

TABLE 8: ARRIVAL_INF STRUCTURE (CONTINUED)

Type	Name	Description of Structure and Data Members
`double`	*delamp*	measured amplitude uncertainty
`double`	*ml_amp*	amplitude from the **amplitude** table (*amptype* = `SBSNR`)
`double`	*ml_snr*	signal-to-noise ratio (snr) from **amplitude.***amp-type* = `SBSNR`
`double`	*per*	measured period
`double`	*ml_per*	measured period for M_L
`double`	*logat*	measured log amplitude/period
`double`	*apma_snr*	*snr* from **apma** table
`double`	*apma_hvrat*	horizontal to vertical ratio from **apma** table
`double`	*apma_rect*	rectilinearity from **apma** table
`double`	*weight*	weight of observation based on user-defined coefficients; each of the time, azimuth, and slowness data have a coefficient
`double`	*belief*	*belief* attribute from **assoc** table
`Stassid_pt*`	*staspt*	pointer to `stassid` structure, if necessary; `NULL` if no valid *stassid* exists
`int`	*assoc*	flag; if *assoc* = `0`, arrival not associated
`int`	*ml_id*	ID of the **amplitude** record for M_L amplitude measurements; passed to the **stamag** record if the arrival is associated and the amplitude is used in M_L calculations
`int`	*mb_id*	ID of the **amplitude** record for m_b amplitude measurements; passed to the **stamag** record if the arrival is associated and the amplitude is used in m_b calculations
`Bool`	*wc_restricted*	flag; if `TRUE`, do not count in weighted count
`Bool`	*aa_processed*	flag; if `TRUE`, arrival already processed
`Bool`	*probdet_restricted*	flag; if `TRUE` do not take into account for probability of detection

TABLE 8: ARRIVAL_INF STRUCTURE (CONTINUED)

Type	Name	Description of Structure and Data Members
`Bool`	*locked_association*	flag; if `TRUE` do not disassociate
`Bool`	*extracted_assoc*	flag; If `TRUE` do not predict
`Bool`	*requested*	flag; requested arrival
`Bool`	*analyst_reviewed*	flag; analyst-reviewed arrival
`Bool`	*driver_restricted*	flag; if `TRUE`, the arrival is not used as a driver arrival
`int`	*drl_count*	driver (event) count associated with this arrival (detection)
`Dr_list*`	*drl_anchor*	pointer to the first `Driver` in the linked list of `Drivers` to which this arrival is associated
`Dr_list*`	*drl_current*	pointer to current `Dr_list` structure

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. For serious errors captured in return, such as a memory allocation error, the process writes an informative error message, then exits; for others, a warning message is written to the log file.

Extract Large Events

The primary purpose of the *Extract Large Events* process is to reduce the overall processing time when the data contain one or more large events that have a large number of detections. The presence of a large event in a time interval usually causes *GAassoc* to form a large number of preliminary event hypotheses, each containing a subset of the arrivals from the large event.

The *Extract Large Events* process is designed to identify and extract large events from a list of preliminary event hypotheses and to resolve the conflicts with events containing a subset of its arrivals at an early stage in the processing. Conflicts are resolved in favor of the large event, with the result that the total number of event hypotheses is considerably reduced and the subsequent processing time is similarly reduced.

The following system requirement is addressed by the *Extract Large Events* process:

- The process shall be efficient enough to form events reflecting natural variations in seismicity in real time.

Input/Processing/Output

The *Extract Large Events* process uses the linked list of `Driver` structures and the GA parameters as input. The algorithm used to extract large events is explained in [IDC5.2.1]. This paragraph presents an overview of the algorithm. Figure 8 shows a detailed data flow and control flow of the process. A large event is identified by the number of its defining phases. The threshold for the minimum number of defining phases of a large event is set by a user parameter. After a large event is identified, it undergoes a split analysis to remove any degenerate cases. Degenerate cases include instances of two or more arrivals at the same station or the same arrival identified as two different phases. These conditions are allowed previous to this stage of *GAassoc* processing. After the *Split Analysis* process, the event is located and confirmed using the *Locate and Confirm Preliminary Event Hypotheses* process. Finally, all conflicts existing for the arrivals of the large event are resolved in favor of the large event. This is a recursive algorithm, where the largest event in the set is processed first. The output of the *Extract Large Events* process is a linked list of `Driver` structures.

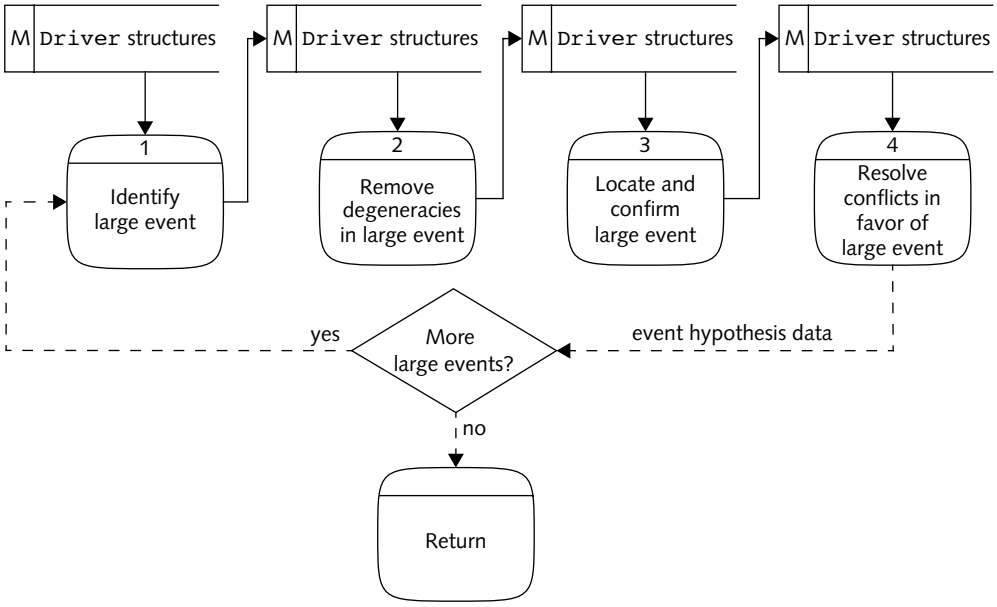


FIGURE 8. EXTRACT LARGE EVENTS PROCESSING SEQUENCE

Control

The *Extract Large Events* process is modularized within one single subroutine, `GA_extirp_large()`, called from within the main *GAassoc* program. After execution of the process, control returns to the main program.

Interfaces

The primary data structure exchanged between the different modules of the *Extract Large Events* process is the linked list of `Driver` structures. The linked list is passed as a handle between the different modules, as well as between the main program and this process.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. For this particular process, a negative return code is passed to the main program if a memory allocation error is encountered.

Eliminate Redundant Events

The purpose of the *Eliminate Redundant Events* process is to reduce the number of preliminary event hypotheses formed at the exhaustive search stage by eliminating redundancies and thus to reduce the overall processing time in *GAassoc*. The following system requirement leads to the design of the *Eliminate Redundant Events* process:

- The process shall form events in a timely manner.

The very existence of this process is based on this requirement. Without *Eliminate Redundant Events*, it is unlikely that GA processing would be efficient enough to satisfy the real-time processing constraint.

Figure 6 on page 27 shows two instances of *Eliminate Redundant Events* (7a and 7b). This non-conventional data flow diagram illustrates that the process is used twice within the *GAassoc* software unit, once before and once after the *Locate and Confirm Preliminary Event Hypotheses* process. It is used with different parameters at these two stages. After the *Locate and Confirm Preliminary Event Hypotheses*, the condition that the driver arrivals be the same for two `Driver` structures to be considered redundant is relaxed.

Input/Processing/Output

Inputs to the *Eliminate Redundant Events* process are the linked list of `Driver` structures and the GA control parameters. During processing each `Driver` is examined once to determine whether or not it is a subset of another `Driver`. For a `Driver` to be a subset of another `Driver`, it must be made up of the same arrivals identified as the same phases. An additional constraint for redundancy, which is

that the same arrival be the driver arrival of both the current `Driver` and its superset, can be imposed. An input parameter controls whether or not this constraint is applied. The constraint is applied the first time that the *Eliminate Redundant Events* process is called, before the *Locate and Confirm Preliminary Event Hypotheses* process. However, it is dropped the second time it is called, after the *Locate and Confirm Preliminary Event Hypotheses* process. Figure 9 illustrates the relationship between the `Driver` and `Arrival_Inf` data structures, with the pointer to the linked list of `Dr_list` data structures pointing back to the `Driver` structure to which the arrival belongs. The `Dr_list` data structure, which is defined in Table 9, is a linked list of pointers to `Driver` structures. The output of this process is a linked list of `Driver` structures where the redundant event hypotheses represented by the structures have been eliminated.

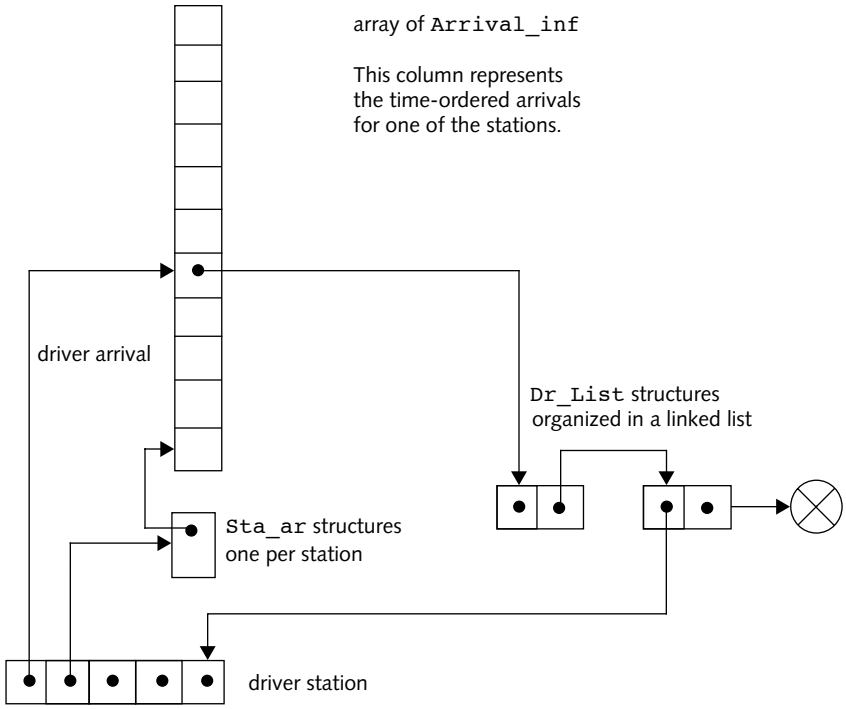


FIGURE 9. DOUBLE-LINK RELATIONSHIP BETWEEN DRIVER STRUCTURES AND ARRIVAL_INF STRUCTURES

TABLE 9: DR_LIST STRUCTURE

Type	Name	Description of Structure and Data Members
`Driver*`	*dr*	pointer to current `Driver`
`Dr_list*`	*prev*	pointer to previous `Dr_list` structure
`Dr_list*`	*next*	pointer to next `Dr_list` structure

Control

The *Eliminate Redundant Events* process is modularized within one function. Control is passed back to the main *GAassoc* program when the process has executed.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process.

Locate and Confirm Preliminary Event Hypotheses

The purpose of the *Locate and Confirm Preliminary Event Hypotheses* process is to locate the event hypotheses and to make sure that they satisfy the confirmation criteria for the following tests:

- weighted-count test

  This test guarantees that events have a minimum number of defining observations established by a weighted-count threshold. The weighted count is a sum of all defining observations where each observation (time, slowness, azimuth) is weighted with a user-defined coefficient.

- arrival-quality test

  This test guarantees that event arrivals have sufficiently well-defined slowness vectors.

- outliers test

  This test guarantees that events do not have associated arrivals with unacceptably large residuals.

A more extensive description of each of these tests is given in [IDC5.2.1]. This section describes the functional aspects of the *Locate and Confirm Preliminary Event Hypotheses* process.

The following system requirements are satisfied by the *Locate and Confirm Preliminary Event Hypotheses* process:

- The subsystem shall produce an automatic bulletin compliant with the database schema [IDC5.1.1Rev2].
- The subsystem shall produce events that satisfy the weighted-count and will have no outliers for any of the time and slowness attributes.
- The subsystem shall produce an acceptable false alarm rate.

Input/Processing/Output

The inputs to the *Locate and Confirm Preliminary Event Hypotheses* process are the `Driver` structures and the parameters that control the location and event confirmation modules. The processing applied sends the association set to the locator function of the *libloc* library [Bra88] and applies the event definition tests mentioned in the preceding section to the event hypotheses. The output of the process is a linked list of `Driver` structures where the preliminary event hypotheses are located and pass all the event definition tests.

Control

The *Locate and Confirm Preliminary Event Hypotheses* process is started when the main *GAassoc* program invokes it. This process is modularized in a single subroutine, `GA_locate_list()`, which is called by the main program, and attempts to locate all preliminary event hypotheses in the list. Within `GA_locate_list()`, the `GA_locate()` subroutine locates a single event and may be called several times

for the same event with different control parameters, for instance with fixed-depth or free-depth conditions. The sequential and iterative calls to the `GA_locate()` function are described in more detail in [IDC5.2.1].

Interfaces

This process interfaces with the *libloc* library at the level of the `GA_locate()` function.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process.

Resolve Conflicts

The *Resolve Conflicts* process is the final algorithmic step in *GAassoc*. The purpose of the process is to produce a final list of event hypotheses where all associated arrivals are associated to one and only one event. The input to the process is a list of event hypotheses that have undergone location, but where arrivals may be associated to several event hypotheses. A detailed explanation of the heuristic approach to resolving the conflicts is given in [IDC5.2.1]. The following system requirements are met by this process:

- The bulletin produced by GA shall be free of conflicting associations.
- The *Resolve Conflicts* process shall be sufficiently efficient to meet timeliness requirements.

Input/Processing/Output

The inputs to *Resolve Conflicts* are the linked list of `Driver` structures and the GA control parameters. The process that leads to a conflict-free bulletin is an iterative process where events are ranked according to their size and quality and processed

in order of decreasing size. A detailed description of the conflict resolution algorithm is given in [IDC5.2.1]. The output of the process is a linked list of `Driver` structures, similar to the input linked list.

Control

The *Resolve Conflicts* process is started when the *GAassoc* program is started. The functions `GA_cluster()` and `GA_assoc_based_CR()` are called directly from the main *GAassoc* program. Control is returned to the main program when the functions have performed their task.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. When a severe error is encountered, an informative error message is written to the log file and the program exits. For non-fatal errors, a message is written to the log file but execution is allowed to proceed uninterrupted.

Write Event Hypotheses to Database

The purpose of the *Write Event Hypotheses to Database* process is to prepare the data in the internal `Driver` structures for output to the database tables constituting the automatic bulletin. The basic system requirement satisfied by this process is that GA produce an automatic bulletin compliant with the database schema [IDC5.1.1Rev2].

Input/Processing/Output

The inputs to the *Write Event Hypotheses to Database* process are the linked list of `Driver` structures and the GA control parameters. The processing recasts the information about the event hypotheses contained in the `Driver` structures in a format intelligible to the *libgdi* library. The outputs of the process are the **origin**, **origerr**, and **assoc** temporary database table entries.

Control

The *Write Event Hypotheses to Database* process is started when the *GAassoc* program is started. This process is not modularized in a function, but rather is part of the task performed by the main *GAassoc* program. This design keeps the interface to the database at the level of the main program.

Interfaces

The *Write Event Hypotheses to Database* process populates the database tables from the `Driver` structures. It uses the *libgdi* library as an interface between *GAassoc* and the relational database management system (RDBMS).

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. In this particular case, errors resulting from calling the *libgdi* functions are captured, the error-reporting functions from the *libgdi* library are called, and the message is written to the log file. The program then exits after calling the rollback function of *libgdi* and closing the database connection using the appropriate *libgdi* function.

DATABASE DESCRIPTION

GAassoc interacts with the database through the Generic Database Interface (GDI). The main interactions of *GAassoc* with the database are at the input stage, where the arrivals from the appropriate networks are read, and at the output stage where the automatic bulletin is written to temporary database tables.

Database Design

GAassoc uses the database for reading the arrival data written by Station Processing, for reading static station and network data, and for writing temporary bulletin tables. The entity-relationship diagram of the schema used by GA is shown in Fig-

ure 4 on page 19. The figure shows all of the tables used by GA. The relationships shown reflect the specific relationships in the context of GA. For instance, the relationship between **origin** and **origerr** is shown as one-to-one when, in general, this is a one-to-zero relationship [IDC5.1.1Rev2]. For *GAassoc* specifically, the **origin**, **origerr**, and **assoc** tables are named **origin_ga_temp**, **origerr_ga_temp**, and **assoc_ga_temp**, as indicated on the diagram. *GAassoc* accesses most of the tables shown on the entity-relationship diagram, except for the **event**, **netmag**, and **stamag** tables, which are accessed only by *GAconflict*.

Database Schema

Table 10 shows the usage of database tables by *GAassoc*. For each table used, the third column shows the purpose for reading or writing each attribute.

TABLE 10: GAASSOC DATABASE USAGE

Table	Action	Usage
site	reads	• *sta*, *ondate*, and *offdate* for record identification • *lat*, *lon*, and *elev* for location information
siteaux	reads	• *sta* for record identification • *nois*, *noissd*, *rely*, and *snthrsh* for detection probability calculation
affiliation	reads	• *sta* for record identification • *net* for identifying auxiliary network, or station elements for creating station intervals
arrival	reads	• *sta* and *arid* for record identification • *time* to relate the arrival to the origin time of a potential event and help in location • *iphase* to identify the arrival's initial phase • *deltim* to take the timing error into account • *azimuth*, *delaz*, *slow*, and *delslo* to use in constraining associations and in location • *stassid* to identify the arrival as belonging to a local or regional group at one station

TABLE 10: GAASSOC DATABASE USAGE (CONTINUED)

Table	Action	Usage
amplitude	reads	• *arid*, *ampid*, *sta*, and *chan* for record identification • *amp*, *snr*, and *per* to use in magnitude computation • *amptype* to identify the type of record (either from the `STA/LTA` or the `A5/2` measurement)
apma	reads	• *arid* for record identification • *snr*, *hvrat*, and *rect* for use in the restricted shear phase test
ga_tag	reads/ writes	• ID for record identification • *state* to identify the tag associated to the arrival
hydro_assoc	reads	• *arid* and *hydro_id* for record identification
hydro_arr_group	reads	• *arid* and *hydro_id* for record identification • *az1*, *az2*, *nhydarr*, *delaz*, and *hyd_grp_phase* to constrain the association of hydroacoustic groups to an event
assoc_ga_temp	writes	• complete records for an event hypothesis as populated by the locator library (*libloc*); this table is used to pass information to *GAconflict*
origin_ga_temp	writes	• complete record for an event hypothesis as populated by the locator library (*libloc*); this table is used to pass information to *GAconflict*
origerr_ga_temp	writes	• complete record for an event hypothesis as populated by the locator library (*libloc*); this table is used to pass information to *GAconflict*

Chapter 4: Detailed Design of GAconflict

This chapter describes the detailed design of *GAconflict* and includes the following topics:

- Data Flow Model
- Processing Units
- Database Description

Chapter 4: Detailed Design of GAconflict

DATA FLOW MODEL

GAconflict is a program designed to resolve conflicts between event association sets formed in successive time segments, as well as between different sectors of the earth (if *GAassoc* has been configured to run as separate instances). In normal automatic processing, *GAconflict* is run immediately after *GAassoc* and is the program that produces the automatic bulletin for a given pipeline. In addition to its primary mission in resolving conflicts, *GAconflict* predicts and associates defining and nondefining phases to complete the formation of event hypotheses. *GAconflict* can also read bulletin data from the results of a previous processing pipeline; for instance, the results of the SEL2 pipeline can be read by the *GAconflict* instance in the SEL3 pipeline.

Figure 10 shows the data flow of *GAconflict* including the main processing units as well as the data stores used as input and output by these processing units. As in the case of *GAassoc*, the internal `Driver` structures serve as a standard data representation for event hypotheses. A string of processes from *Read Event Information* to *Write Event Hypotheses to Database* use this basic data structure as I/O, allowing for a modular design. All of the processes shown on Figure 10 take the list of GA parameters as input. The GA parameters are read by a single process, which is shared with *GAassoc*.

Table 11 on page 60 shows the sequence of processes in *GAconflict* corresponding to Figure 10. The process numbers in the table map to those in the figure.

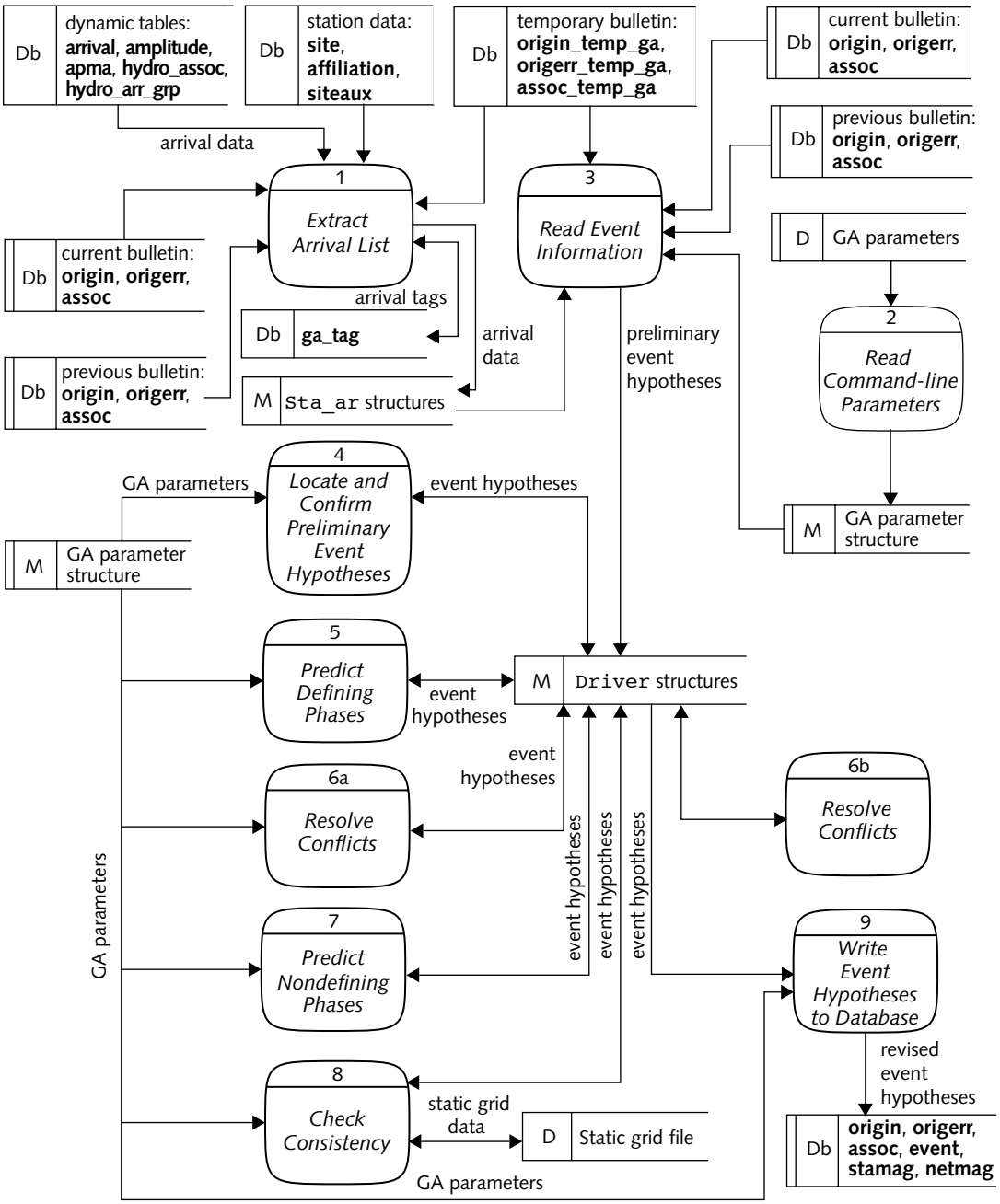


FIGURE 10. GACONFLICT DATA FLOW

TABLE 11: PROCESSING UNITS AND CORRESPONDING C FUNCTIONS

Processing Unit	Process Number on Data Flow	Function Name

Extract Arrival List	1	`GAconflict()`
		`GA_build_arrival_query()`
		`GAarrivals()`

Read Command-line Parameters	2	`GA_read_par()`
Read Event Information	3	`GAconflict()`
Locate and Confirm Preliminary Event Hypotheses	4	`GA_locate_list()` `GA_locate()`
Predict Defining Phases	5	`GA_predict_def()`
Resolve Conflicts	6	`GA_cluster()` `GA_assoc_based_CR()`
Predict Nondefining Phases	7	`GA_predict()`

Check Consistency	8	`GAconflict()`
		`GA_coda_name_check()`
		`GA_check_HT_pairs()`
		`GA_check_depth()`
		`GA_check_range()`
		`GA_rm_incomplete_hydro_groups()`
		`GA_check_predicted_time_residuals()`
		`GA_check_regional_pairs()`
		`GA_check_isolated_secondary()`
		`GA_check_duplicate()`
		`GA_check_max_magnitude_diff()`
		`GA_check_small_deep_events()`
		`GA_check_phase_order ()`
		`GA_check_mag_def()`

Write Event Hypotheses to Database	9	`GAconflict()`

PROCESSING UNITS

GAconflict consists of the following processes, as shown in Figure 10 and listed in Table 11:

- *Extract Arrival List* (1)

  This process reads station and arrival data from the database to produce the `Sta_ar` structures, an internal representation of the arrivals for the time interval being processed.

- *Read Command-line Parameters* (2)

  This process reads command-line parameters and parameters in parameter files and stores the values internally as GA control parameters.

- *Read Event Information* (3)

  This process reads from the various sets of bulletin tables and stores the information in internal `Driver` structures.

- *Locate and Confirm Preliminary Event Hypotheses* (4)

  This process estimates an event's origin time, latitude, longitude, and depth based on the travel-time and slowness attributes of the detections in its association set. Confirmation is the process that verifies that the events meet the required criteria including a weighted count of arrival attributes, an arrival-quality test, and a probability-of-detection test. The weighted-count test compares a weighted sum of arrival attributes against a threshold value. The arrival-quality test compares the sum of the arrival-quality functions over all defining arrivals in the event hypothesis with a threshold value. The arrival-quality function is a rational function of the uncertainty in the measure of slowness and the distance between the event and the station. It is empirical in nature and has been found to help in reducing the number of false event hypotheses constructed from stochastically consistent sets of arrivals.

- *Predict Defining Phases* (5)

  This process predicts defining phases specified by the GA control parameter that lists the defining phases. The prediction is based on the location of the events and takes into account the error ellipse and the uncertainty

in the attributes (time, azimuth, and slowness) of the arrival. The algorithm predicts phases that are not yet associated and conducts an exhaustive search for all possible associations. The best association is picked out of the possible associations based on a chi-square test.

- *Resolve Conflicts* (6)

  This process eliminates the ambiguity and inconsistency of associating an arrival to multiple events. After this process, an arrival is associated to no more than one event.

- *Predict Nondefining Phases* (7)

  This process predicts nondefining phases specified by a GA control parameter listing the phases to be predicted. The prediction is based on the location of the events, but in contrast with the *Predict Defining Phases* process, does not use the error ellipse. However, the uncertainty in the attributes (time, azimuth, and slowness) of the arrival are taken into account. The algorithm predicts phases that are not yet associated and conducts an exhaustive search for all possible associations. The best association is picked out of the possible associations based on a chi-square test.

- *Check Consistency* (8)

  Before the event hypothesis is written to the database, a number of geophysical checks are performed to ensure consistency and adherence to event acceptance criteria.

- *Write Event Hypotheses to Database* (9)

  This process casts the internal representation of the event hypotheses into their external representation in the database using tables **origin**, **origerr**, **assoc**, **event**, **netmag**, and **stamag**.

The following paragraphs describe the design of these processes.

Extract Arrival List

Similar to the process with the same name in *GAcons*, the purpose of this process is to build and execute the queries that read the correct list of arrivals from the database and insert them into internal structures for further processing. The process uses standard system libraries (for example, *libgdi*) to read the arrivals produced by Station Processing from the database based on the time interval to be processed and the list of events to be read for this *GAconflict* session. This list of events is established by the *Read Event Information* process, which is specific to *GAconflict*.

The network names and the number of networks for each technology are flexible, and the module that builds the arrival query is re-usable in other programs.

Input/Processing/Output

The inputs to this process are GA control parameters read from a parameter file, including the start- and end-time, static database tables (**affiliation**, **site**, and **siteaux**), dynamic database tables (**arrival, amplitude, apma, hydro_assoc**, and **hydro_arr_group**), as well as the tables that describe events in the temporary bulletin (**origin_temp_ga**, **origerr_temp_ga**, and **assoc_temp_ga)**, the current bulletin (**origin**, **origerr**, and **assoc**), and the previous bulletin (**origin**, **origerr**, and **assoc**), as indicated on Figure 10 on page 59. The processing populates the `Sta_ar` data structures and updates the **ga_tag** database table, which are the outputs of this unit.

Control

The *Build Static Grid* process is started when *GAconflict* is started. When severe error conditions are encountered, for instance with database input, the program writes an informative message to standard output, which is usually directed to the log file, then exits.

Interfaces

The *Extract Arrival List* process uses the *libgdi* library to acquire its database input. All of the database interface functions from *libgdi* are called in the main `GAconflict()` program. `GA_build_arrival_query()` builds a query string based on the input control parameter for the current interval and the list of events that should be read for this interval. The query string is passed to the *libgdi* library for execution. Finally, the results of the query are parsed by `GAarrivals()`, which also populates the `Sta_ar` internal structures.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. In this particular case, the process is mostly a series of calls to the *libgdi* subroutines. After these calls are made, the *libgdi* error handling subroutine is called, any error messages are printed to the log file, the database connection is closed, and the process exits.

Read Command-line Parameters

Read Command-line Parameters is used both by *GAassoc* and *GAconflict*. Refer to "Read Command-line Parameters" on page 31 for a description of this process.

Read Event Information

This process selects the appropriate GA event hypotheses from the temporary tables written by *GAassoc*, from the final bulletin tables for the current pipeline, and from the GA bulletin of a previous pipeline.

Input/Processing/Output

The inputs to this process are the temporary bulletin database tables written by *GAassoc,* the bulletin tables that contain the final bulletin, the bulletin tables from a previous pipeline, and the `Sta_ar` structures. The processing reads the appropriate events from the database tables and sets up, as output, the linked list of `Driver` structures for further processing.

Control

The *Read Event Information* process is an integral part of the *GAconflict* main program and is activated when the program is activated. Control passes to the *libgdi* functions to interface with the database, and the process exits when an error condition is encountered within the *libgdi* functions.

Interfaces

The *Read Event Information* process interfaces with the *libgdi* library at the level of the *GAconflict* main program.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. In this particular case, the process is mostly a series of calls to the *libgdi* subroutines. After these calls are made, the *libgdi* error handling subroutine is called, any error messages are printed to the log file, the database connection is closed, and the process exits.

Locate and Confirm Preliminary Event Hypotheses

This is the same process as in *GAassoc*. Refer to "Locate and Confirm Preliminary Event Hypotheses" on page 49 for a description of this process.

Predict Defining Phases

The *Predict Defining Phases* process completes events that have been located and confirmed with additional, previously non-associated defining detections. This step is necessary because the automatic association process may remove detections from the association set at some steps of the algorithm, such as *Resolve Conflicts* or outlier analysis within the *Locate and Confirm Preliminary Event Hypotheses* process. The *Predict Defining Phases* process provides a mechanism for reassociating arrivals consistent with the event hypotheses. The following algorithmic steps are taken within the *Predict Defining Phases* process:

- Predict defining phases at stations that are not already associated.
- Use the error ellipse when predicting time and slowness attributes.
- Relocate the event, and perform an outlier analysis if one or more defining phases were added.

Input/Processing/Output

The *Predict Defining Phases* process takes the linked list of `Driver` structures and the GA parameters as input. For each event, the time, azimuth, and slowness attributes of defining phases are predicted at stations that are not associated to the event. The error ellipse, model errors, and measurement errors are taken into account when predicting the time and slowness for the defining phases. The output of this process is a linked list of `Driver` structures. The algorithm used for the prediction is described in detail in [IDC5.2.1].

Control

The *Predict Defining Phases* process is modularized in a single function, which is called from the main *GAconflict* program.

Interfaces

The *Predict Defining Phases* process, like most processes within *GAassoc* and *GAconflict*, uses the `Driver` structure as its main argument and transforms this structure by adding predicted arrivals to the event hypotheses. In addition to the `Driver` linked list, the process takes the GA parameters as arguments.

A special structure, called the `Pred_triplet`, is defined and used in this process and in the *Predict Nondefining Phases* process. It is used to decide the arrival that best fits the current event and phase out of a set of several arrivals. This structure is shown in Table 12.

TABLE 12: PRED_TRIPLET STRUCTURE

Type	Name	Description of Structure and Data Members
`Sta_Ar*`	*stpt*	pointer to `Sta_Ar` structure for this arrival
`char*`	*sta*	pointer to station name
`char*`	*phase*	pointer to phase name
`Arrival_Inf*`	*ar*	pointer to arrival
`double`	*fit*	chi-square fit of this arrival to current event
`double`	*timeres*	time residual for this arrival
`double`	*azres*	azimuth residual
`double`	*slores*	slowness residual
`double`	*delta*	distance between station and event
`double`	*seaz*	station to event azimuth
`double`	*esaz*	event to station azimuth
`Pred_triplet*`	*next*	pointer to the next `Pred_triplet` structure

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. In this particular case, an error code is returned to the main *GAconflict* program when an error condition is encountered. An informative error message is written to the log file before control is returned to the main program.

Resolve Conflicts

This process is identical to the *Resolve Conflicts* process in *GAassoc*. Refer to "Resolve Conflicts" on page 51 for a description of this process.

Predict Nondefining Phases

The purpose of the *Predict Nondefining Phases* process is to supplement existing events with nondefining, usually late-arriving phases whose list is specified by a parameter.

Input/Processing/Output

The inputs to the *Predict Nondefining Phases* process are the linked list of `Driver` structures and the GA control parameters. For each event, the time, azimuth, and slowness attributes of nondefining phases are predicted at stations that already possess defining, primary phases. Model and measurement errors are used when predicting the time and slowness for the predicted defining phases. The following algorithmic steps are taken within the *Predict Nondefining Phases* process:

- Predict nondefining phases at stations that already have primary defining phases associated.
- Compute the time and slowness windows for the predicted arrivals using measurement and modeling errors for time and slowness.

The prediction algorithm is explained in more detail in [IDC5.2.1]. The output of this process is the linked list of `Driver` structures.

Control

The *Predict Nondefining Phases* process is an integral part of the *GAconflict* program and is activated when the program is activated.

Interfaces

The *Predict Nondefining Phases* process, like most processes within *GAassoc* and *GAconflict*, uses the `Driver` structure as its main argument and modifies this structure by adding predicted arrivals to the event hypotheses. In addition to the `Driver` linked list, the process uses the GA parameters. The `Pred_triplet` structure is designed for use within this process and the *Predict Defining Phases* process and is instrumental in deciding the arrival that best fits the current event and phase out of a set of several arrivals. This structure is presented in Table 12 on page 67.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process.

Check Consistency

The purpose of the *Check Consistency* process is to make sure that the event hypotheses generated by *GAconflict* satisfy a number of criteria before they are written to the database. The following criteria are described in detail in [IDC5.2.1]:

- Coda phase names are valid.
- Each station can have no more than one hydroacoustic phase.
- Deep events occur in zones of deep seismicity.
- Distance and depth ranges are valid.
- Hydroacoustic groups are complete.
- Residuals of nondefining phases are screened.
- Regional phase pairs are valid.

- Secondary phases are not isolated (if the corresponding parameter is set).
- Phases are not duplicated.
- Local magnitude is screened for being an outlier.
- Deep events are not small (unless the solution is invalid at the surface).
- Phases are in order.
- Outlier station magnitudes are nondefining.

Input/Processing/Output

The inputs to *Check Consistency* are the GA control parameters and the linked list of `Driver` structures representing the event hypotheses. A specific software module enforces each of the criteria listed in the previous section, and every event is passed through the suite of modules. When an event fails one of the criteria and is modified accordingly (for instance, one of the phases is renamed to obtain a valid regional phases pair, or a depth is attempted at the surface for small deep events), it is resubmitted to the whole suite of tests. The output of this process is the linked list of `Driver` structures.

Control

The *Check Consistency* process is a series of calls to the functions specified in Table 11 on page 60. These calls are made from the main *GAconflict* program, and control is returned to that main program when the functions have finished executing. Action is taken in the main program in case an event must be removed, as indicated from the return flag from one of the functions called. The `GA_rm_event_pt_to_next()` function is then called from the main program.

Interfaces

Each event in the linked list of events is passed by reference from one module checking on a specific criterion to the next module. If an event is modified to comply with one of the criterion after it fails a test, it is re-submitted to the whole suite of tests.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process.

Write Event Hypotheses to Database

This process is very similar to the process with the same name from the *GAassoc* software unit (see "Write Event Hypotheses to Database" on page 52). The only difference is that **event**, **netmag**, and **stamag** database table entries are written to the database in addition to the **origin**, **origerr**, and **assoc** table entries.

DATABASE DESCRIPTION

GAconflict interacts with the database through the *libgdi* software library. The main interactions between *GAconflict* and the database are at the input and output stages. At the input stage, the arrivals from the appropriate networks are read as well as the event hypotheses from the temporary, previous, and current database tables. At the output stage, the event hypotheses are written out to the current database tables.

Database Design

GAconflict uses the database for reading input data and static tables and for writing output to the final bulletin tables. The entity-relationship diagram of the schema is shown in Figure 4 on page 19. All of the tables shown on this entity-relationship diagram are accessed by the *GAconflict* software unit. The relationships shown reflect the specific relationships in the context of GA. *GAconflict* accesses three sets of bulletin tables (**origin**, **origerr**, and **assoc**): the temporary GA tables (temporary in the sense that the data they contain are local to GA and are cleared prior to each execution of the sequence *GA_DBI*, *GAassoc*, then *GAconflict*), the previous-pipeline bulletin tables (which are accessed for reading only), and the current-pipeline bulletin tables (which are accessed both for reading and

writing). Figure 11 shows the relationship of *GAassoc* and *GAconflict* to the various sets of bulletin tables. The temporary tables are the only bulletin tables accessed by *GAassoc*. *GAconflict*, however, reads the temporary tables, reads the previous bulletin tables, and both reads and writes the current bulletin tables.

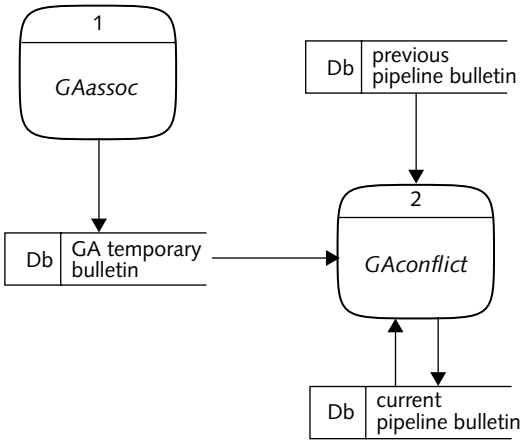


FIGURE 11. INPUT AND OUTPUT BULLETIN TABLES IN GAASSOC AND GACONFLICT

Database Schema

Table 13 shows the usage of database tables by *GAconflict*. For each table used, the third column shows the purpose for reading or writing each attribute.

TABLE 13: GACONFLICT DATABASE USAGE

Table	Action	Usage
site	reads	• *sta*, *ondate*, and *offdate* for record identification • *lat*, *lon*, and *elev* for location information
siteaux	reads	• *sta* for record identification • *nois*, *noissd*, *rely*, and *snthrsh* for detection probability
affiliation	reads	• *sta* for record identification • *net* for identifying auxiliary network or station elements for creating station intervals
arrival	reads	• *sta and arid* for record identification • *time* to relate the arrival to the origin time of a potential event and help in location • *iphase* to identify the arrival's initial phase • *deltim* to take the timing error into account • *azimuth*, *delaz*, *slow*, and *delslo* to use in constraining associations and in location • *stassid* to identify the arrival as belonging to a local or regional group at one station
amplitude	reads	• *arid*, *ampid*, *sta*, and *chan* for record identification • *amp*, *snr*, and *per* to use in magnitude computation • *amptype* to identify the type of record (either from the `STA/LTA` or the `A5/2` measurement)
apma	reads	• *arid* for record identification • *snr*, *hvrat*, and *rect* for use in the restricted shear phase test
ga_tag	reads/ writes	• *ID* for record identification • *state* to identify the tag associated to the arrival
hydro_assoc	reads	• *arid* and *hydro_id* for record identification

TABLE 13: GACONFLICT DATABASE USAGE (CONTINUED)

Table	Action	Usage
hydro_arr_group	reads	• *arid* and *hydro_id* for record identification • *az1*, *az2*, *nhydarr*, *delaz*, and *hyd_grp_phase* to constrain the association of hydroacoustic groups to an event
assoc (from current pipeline)	reads/ writes	• complete record as populated by the locator library
origin (from current pipeline)	reads/ writes	• complete record as populated by the locator library
origerr (from current pipeline)	reads/ writes	• complete record as populated by the locator library
assoc (from previous pipeline)	reads	• complete record as populated by the locator library
origin (from previous pipeline)	reads	• complete record as populated by the locator library
origerr (from previous pipeline)	reads	• complete record as populated by the locator library
event	writes	• complete record indicating the preferred origin record for each event
stamag	writes	• complete record of station magnitude for all stations where it is available
netmag	writes	• complete record of network magnitude for all events where it is available
assoc_ga_temp	reads	• complete records for an event hypothesis as populated by the locator library (*libloc*)
origin_ga_temp	reads	• complete record for an event hypothesis as populated by the locator library (*libloc*)
origerr_ga_temp	reads	• complete record for an event hypothesis as populated by the locator library (*libloc*)

Chapter 5: Detailed Design of GA_DBI

This chapter describes the detailed design of *GA_DBI* and includes the following topics:

- Data Flow Model
- Processing Units
- Database Description

Chapter 5: Detailed Design of GA_DBI

DATA FLOW MODEL

The *GA_DBI* program is used in the automatic pipeline to perform special-purpose tagging of arrivals to support specialized processing by *GAassoc* and *GAconflict*. Examples of arrivals that are tagged include arrivals from auxiliary stations and hydroacoustic detections when an overflow situation (large number of defining hydroacoustic detections within a short time period, with no azimuth information) exists. *GA_DBI* is run before *GAassoc* in the automatic pipeline. Figure 12 shows the data flow for *GA_DBI*.

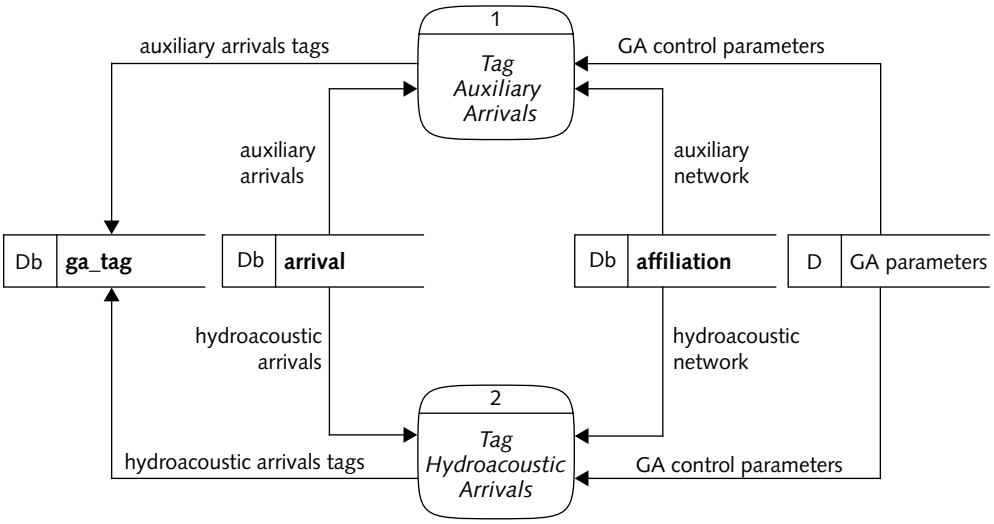


FIGURE 12. GA_DBI DATA FLOW

PROCESSING UNITS

As shown in Figure 12, *GA_DBI* has a simple structure and consists of two processes:

- *Tag Auxiliary Arrivals*
- *Tag Hydroacoustic Arrivals*

The following paragraphs describe the design of these processes.

Tag Auxiliary Arrivals

The purpose of the *Tag Auxiliary Arrivals* process is to label auxiliary seismic arrivals in the set of arrivals loaded by *GAassoc* and *GAconflict* and to insert a record into the **ga_tag** table identifying the arrival as `WC_RESTRICTED`, `REQUESTED`, and `PROBDET_RESTRICTED`. The `WC_RESTRICTED` tag indicates that the arrival does not contribute to the weighted-count test in the *Locate and Confirm Preliminary Event Hypotheses* process. The `REQUESTED` tag indicates that the arrival is from an auxiliary station, and the `PROBDET_ RESTRICTED` tag indicates that the arrival is not used in the probability-of-detection test.

Input/Processing/Output

The inputs to the *Tag Auxiliary Arrivals* process are the set of GA parameters and the **affiliation** and **arrival** database tables. This process identifies the arrivals from auxiliary seismic stations and inserts entries into the **ga_tag** table to identify the auxiliary arrivals as such and tag them with all three tags: `WC_RESTRICTED`, `REQUESTED`, and `PROBDET_RESTRICTED`. The outputs of the *Tag Auxiliary Arrivals* process are the records inserted into the **ga_tag** database table.

Control

The *Tag Auxiliary Arrivals* process is an integral part of the *GA_DBI* program and is activated when the program is activated.

Interface

This process interfaces with database through the *libgdi* library.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. In this particular case, the possible errors resulting from calling the *libgdi* functions are captured, the error-reporting functions from the *libgdi* library are called, and the message is written to the log file. The program exits after calling the rollback function of *libgdi* and closing the database connection using the appropriate *libgdi* function.

Tag Hydroacoustic Arrivals

The purpose of the *Tag Hydroacoustic Arrivals* process is to label time intervals during which hydroacoustic (H) arrivals are detected at a rate above a defined threshold. When such a situation is detected, low-snr defining hydroacoustic arrivals are tagged as `DRIVER_RESTRICTED`.

Input/Processing/Output

The inputs to the *Tag Hydroacoustic Arrivals* process are the GA parameters and the **affiliation** and **arrival** database tables. Processing detects the high detection rate condition (as defined by a threshold parameter of *n* arrivals per hour), inserts entries into the **ga_tag** table to identify the hydroacoustic arrivals as such, and tags the arrivals with the tag `DRIVER_RESTRICTED`. The output of *Tag Hydroacoustic Arrivals* are the records inserted in **ga_tag**.

Control

The *Tag Hydroacoustic Arrivals* process is an integral part of the *GAconflict* program and is activated when the program is activated.

Interface

This process interfaces with the database through the *libgdi* library.

Error States

Defensive programming is used throughout the source code for GA, including this process, by capturing and flagging any error from the functions called within the process. In this particular case, the possible errors resulting from calling the *libgdi* functions are captured, the error-reporting functions from the *libgdi* library are called, and the message is written to the log file. The program exits after calling the rollback function of *libgdi* and closing the database connection using the appropriate *libgdi* function.

DATABASE DESCRIPTION

GA_DBI interacts with the database through the GDI. The main interactions between *GA_DBI* and the database are at the input stage, where the arrivals from the appropriate networks are read, and at the output stage where the arrival tagging is performed.

Database Design

GA_DBI uses the database for reading input data and static database tables and for writing output to the **ga_tag** tables. The entity-relationship model of the schema used by *GA_DBI* is shown in Figure 4 on page 19. The figure shows all of the tables accessed by the GA subsystem in general. Only a fraction of the tables shown on that figure are accessed by *GA_DBI*; these tables are listed in Table 14.

Database Schema

Table 14 shows the usage of database tables by *GA_DBI*. For each table used, the third column shows the purpose for reading or writing each attribute.

TABLE 14: GA_DBI DATABASE USAGE

Table	Action	Usage
affiliation	reads	• *sta* for record identification • *net* for identifying auxiliary network, or station elements for creating station intervals
arrival	reads	• *sta* and *arid* for record identification • *time* to relate the arrival to the origin time of a potential event • *iphase* to identify the arrival's initial phase • *snr* to be used in the *Tag Hydroacoustic Arrivals* process
ga_tag	writes	• *state* for identifying the arrival's tagging • *objtype* for identifying the type of object • *id* for record identification

Chapter 6: Detailed Design of GAcons

This chapter describes the detailed design of *GAcons* and includes the following topics:

- Data Flow Model
- Processing Units
- Database Description

Chapter 6: Detailed Design of GAcons

DATA FLOW MODEL

GAcons is a utility program that precomputes and stores propagation knowledge base information used by *GAassoc*, *GAconflict*, and *GAgrid*. The information is stored in two grid files, the Propagation Knowledge Base grid file and the Static grid file. Precomputing and storing this information is much more efficient than computing it on the fly in the *GAassoc* program. The grid files generated by *GAcons* are written using a flat file format described in the next section. Figure 13 shows the data flow for *GAcons*.

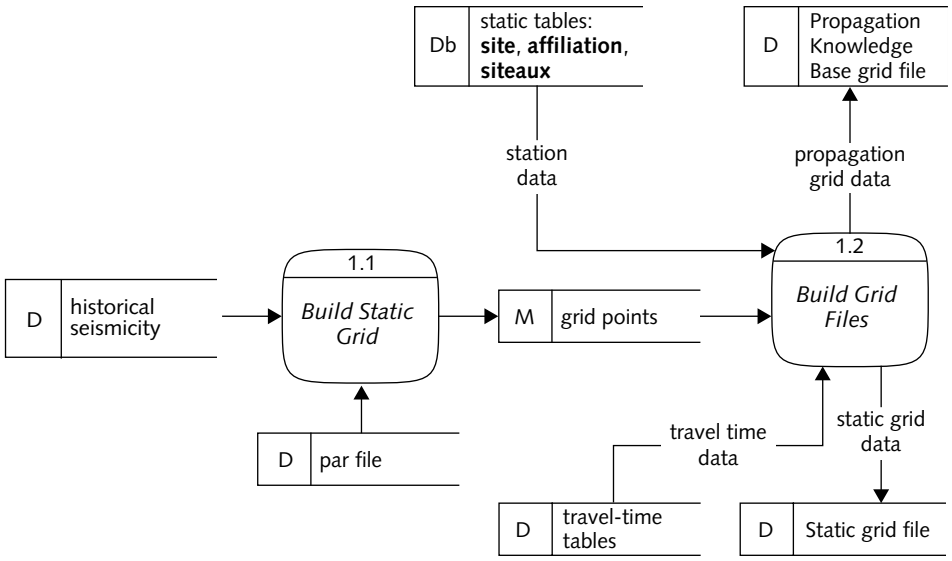


FIGURE 13. GACONS DATA FLOW

PROCESSING UNITS

GAcons consists of the following processes (see Figure 13):

- *Build Static Grid*
- *Build Grid Files*

The following paragraphs describe the design of these processes.

Build Static Grid

The *Build Static Grid* process sets up the coordinates of the global grid points. The algorithm used to define the grid is described in detail in [IDC5.2.1]. The *Build Static Grid* process can be configured for different grid spacing and different deep historical seismicity statistics. Deep seismicity statistics (shown in Figure 13 as the "historical seismicity" data store) are used to establish the location of the grid points at depth. The statistics are a list of events occurring at depth over a long time period, extracted from a standard event bulletin.

Input/Processing/Output

The input to the *Build Static Grid* process is the *GAcons* parameter file, which includes the value of the average grid spacing and the name of the ASCII file containing the deep seismicity statistics used to establish the grid points at depth. The historical seismicity file is a list of seismic events (one line per event) with latitude, longitude, depth, and body-wave magnitude in floating point format. Latitude and longitude are expressed in decimal degrees, the depth in kilometers, and the magnitude in magnitude units.

The processing in the *Build Static Grid* process consists of establishing the static grid over the whole globe with the exact location of each grid file.

The output of the *Build Static Grid* process is the linked list of `Grid_pt` data structures.

Control

The *Build Static Grid* process is the first process to be executed within *GAcons*. It is an integral part of the program and is not controlled independently. Its purpose is to establish the list of grid points used in the exhaustive grid search algorithm by *GAassoc* and to populate the appropriate data structures.

Interfaces

The *Build Static Grid* process populates a linked list of `Grid_pt` data structures. This is a linked list in the sense that each element of the list points to the previous element and the next element within the list. The whole list can be passed to different modules by simply passing the handle to the first element (pointer to the first element pointer) within the list. That handle may also be called the "anchor". Each element of the linked list contains geographical information about the grid cells. Table 15 displays the content of the `Grid_pt` data structure with a short description of the data members of that structure. The `Grid_pt` data structure is defined in the source code in file `libGA.h`.

TABLE 15: GRID_PT STRUCTURE

Type	Name	Description of Structure and Data Members
`long`	*index*	unique beam point identifier number
`float`	*lat*	latitude of current beam point (deg.)
`float`	*lon*	longitude of current beam point (deg.)
`float`	*depth*	depth of mid-point of cell (km)
`float`	*lower_depth_bound*	lower depth bound (km)
`float`	*upper_depth_bound*	upper depth bound
`float`	*radius*	radius of cell around grid point (deg.)
`float`	*b_value*	b value in grid cell
`Grid_pt*`	*next*	pointer to next grid point; NULL if last
`Grid_pt*`	*prev*	pointer to previous grid point; NULL if first

Error States

The *Build Static Grid* process is not a critical component of the operational system in terms of reliability because it is used only occasionally to build the files containing the knowledge base used by *GAassoc* and *GAconflict*. No failure of this component has been observed, and defensive programming is used to capture any error message and inform the user of the failure condition on the standard output. Error messages for this utility are output either on the UNIX `stderr` or on the UNIX `stdout` stream and are captured in a file by re-directing one or both of these standard UNIX outputs to a file. The following example shows a warning message that is written to `stderr` when the name of the seismicity file is not given accurately:

```
GAdepth_build: Couldn't open file:
/cmss/config/earth_specs/GA//1980-1993.pde_depts
--> Depth info will be ignored !!!
```

In this case, *GAcons* continues after writing the message. More severe errors, such as a problem accessing the database, cause the program to exit.

Build Grid Files

The purpose of the *Build Grid Files* process is to populate the data structures containing the travel-time and slowness information for all grid cells and all stations and write that information out to the two output grid files. The locations of the grid points and the size of the cells output by the *Build Static Grid* process are used as a basis to compute the structure contents listed in Tables 18 and 19 on page 90 for each of the grid cells. The structure contents include the propagation characteristics from the cell for each station and seismic or acoustic phase. These propagation characteristics are used by the exhaustive grid search algorithm in *GAassoc* to create new event hypotheses.

Input/Processing/Output

The following are inputs to the *Build Grid Files* process:

- linked list with the grid point structure, `Grid_pt`, containing the location of the grid points as well as the size of the cells
- travel-time tables and applicable corrections [Ken91a]
- station and network information contained in database tables **site**, **siteaux**, and **affiliation**
- *GAcons* control parameters applicable to the *Build Grid Files* process
- magnitude attenuation tables
- file, named `slowamp.P`, used by the probability-of-detection algorithm to compute the minimum magnitude detectable at a given station for an event within the grid cell
- hydroacoustic blockage files
- file used to define the valid distance and depth range of the various seismic, hydroacoustic, and infrasonic phases

The *Build Grid Files* process computes the attributes from the geographical information contained in the grid locations and in the static tables. It then places that information in the data structures described in Tables 18 and 19 on page 90 and writes that information to two separate binary grid files. The two grid files are the principal outputs of the *Build Grid Files* process. To differentiate between the two grid files, they are referred to as the Propagation Knowledge Base grid file and the Static grid file.

The Propagation Knowledge Base grid file contains network-specific information that is used by *GAassoc*. This file allows for a flexible number of grid points, stations, and phases. A record in the Propagation Knowledge Base grid file is called a Beam Point record. A Beam Point record contains all of the information pertaining to the physical point at the center of a cell. Beam Point records are written one after the other to the grid file. At the end of each Beam Point record, a four-byte

integer flag indicates whether or not a Beam Point record follows (Figure 14). A value of one indicates that a record follows, and a value of zero indicates the end of the file.

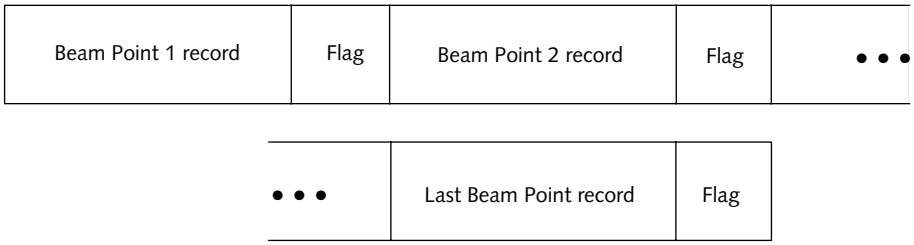


FIGURE 14. PROPAGATION KNOWLEDGE BASE GRID FILE STRUCTURE

The structure of a Beam Point record is shown in Figure 15. The first two elements of the Beam Point record (the header) are the index number of the grid point and the size of the record. Following this header is a Grid Point record containing static information such as latitude, longitude, radius of the cell, and so on. The length of this record is equal to the size of the `grid_pt` structure (`Grid_pt`) as listed in Table 15 on page 84. The next records contain first-arrival station information. For each station that could detect the earliest arrival from an event located in this cell, a First-station record of length equal to the size of the `StaPt` structure is written (see Table 18 on page 90). First-station records are separated by a flag indicating whether or not another First-station record follows the current one. A value of one indicates that more records follow, and a value of zero indicates the last record.

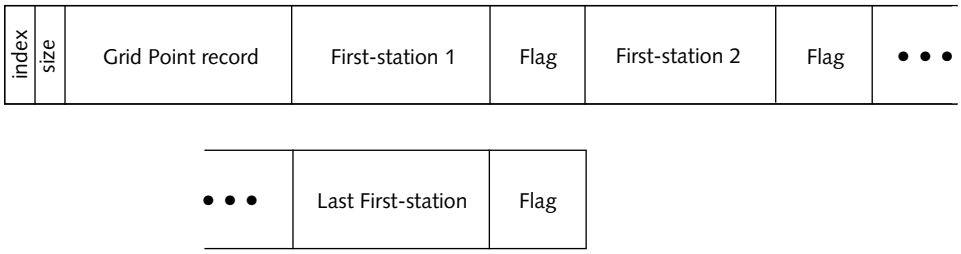


FIGURE 15. BEAM POINT RECORD STRUCTURE WITHIN GRID FILES

Following the last First-station record are structures containing Station records for all stations in the network and Phase records for all phases specified by the control parameters. The structure of Station records (which contain Phase records) is shown in Figure 16. The Station record comes first and is followed by the first Phase record (the length of this record is equal to the size of the `Phas_Inf` C structure; see Table 19 on page 91). All Phase records are written in a sequence separated by flags. The last Phase record for a station is followed by a flag indicating that no more Phase records follow. This flag is followed by another flag indicating the presence (one) or absence (zero) of another station record. Two sequential zero flags indicate the end of the last station for this beam point.

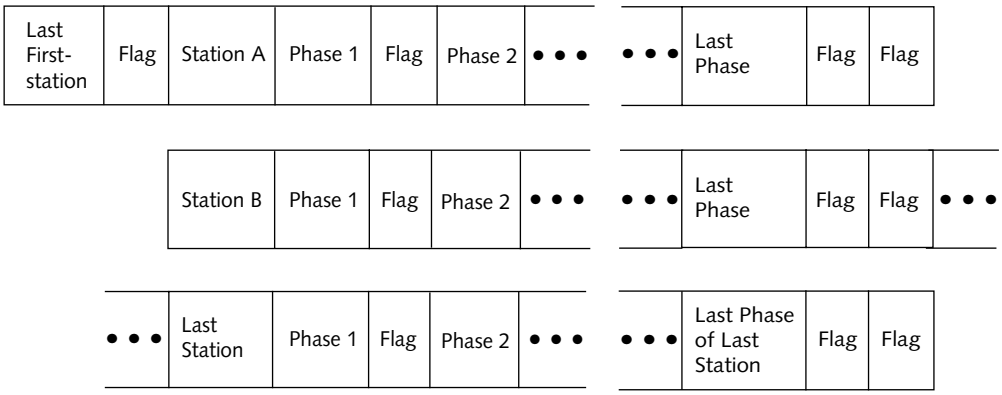


FIGURE 16. STATION RECORD STRUCTURE

The Grid Point record shown in Figure 15 on page 88 contains a `Grid_Pt` structure (see Table 15 on page 84). The last two elements of this structure are ignored when reading the file. They contain the pointers to the previous and next Grid Point records, and they are re-initialized to current values after reading the file.

Each Station record shown in Figure 16 contains attributes characterizing the station-beam-point pair. The format is the same as the `StaPt` C structures shown in Table 18 on page 90. A number of Phase records are attached to each Station record. These records characterize the propagation between the station and the beam point. All phases that are specified by the control parameters and that can exist at the station for an event within the grid cell are included as Phase records. The format of the Phase record follows the definition of the `Phas_Inf` C structure shown in Table 19 on page 91.

Control

The *Build Grid Files* process, like the *Build Static Grid* process, is an integral part of the *GAcons* program and is not controlled independently. It is invoked at the conclusion of the *Build Static Grid* process and when it exits, *GAcons* exits.

Interfaces

The *Build Grid Files* process populates the data structures defined in Tables 16 through 19 using the travel-time information derived from the location of the grid points, the stations, and the travel-time tables. The input data for the *Build Grid Files* process are brought into the system using the standard interface libraries of *libgdi* for the database tables, *libloc* for the travel-time data, and *libmagnitude* for the magnitude data. Figure 17 shows the relationships between the main data structures handled within the *Build Grid Files* process. Each large rectangle in the figure represents a data structure; the smaller rectangles with a dot represent pointers and the ones lacking a dot represent data. The arrows show the entities pointed to by these pointers. The crossed circles represent NULL pointers, which are placed at the end of a linked list.

TABLE 16: BEAM_PT STRUCTURE

Type	Name	Description of Structure and Data Members
`Grid_pt*`	*loc*	pointer to `Grid_pt` structure containing grid location information
`First_Sta*`	*first_sta*	pointer to stations with potential for recording first arrival
`StaPt*`	*stpt*	pointer to linked list of all stations
`Beam_pt*`	*next*	pointer to next `Beam_pt`; NULL if last
`Beam_pt*`	*prev*	pointer to previous `Beam_pt`; NULL if first

TABLE 17: FIRST_STA STRUCTURE

Type	Name	Description of Structure and Data Members
`StaPt*`	*sta_pt*	pointer to station-beam point information
`First_Sta*`	*next*	pointer to next `First_Sta` structure

TABLE 18: STAPT STRUCTURE

Type	Name	Description of Structure and Data Members
`char`	*sta*[GA_STA_NAME]	station code (name)
`int`	*tab_index*	index into **arrival** table
`float`	*delta*	distance from Beam-point center (deg.)
`float`	*azi*	azimuth between beam cell center and station (deg.; angle measured at station)
`float`	*baz*	back-azimuth (deg.; angle measured at beam cell center)
`float`	*min_mag*	minimum detectable magnitude at station
`float`	*mag_cor*	magnitude correction at center of cell
`float`	*d_mag_cor_dr*	radial derivative of magnitude correction

TABLE 18: STAPT STRUCTURE (CONTINUED)

Type	Name	Description of Structure and Data Members
`float`	*d_mag_cor_dz*	vertical derivative of magnitude correction
`Phas_Inf*`	*Ppt*	pointer to phase information
`StaPt*`	*next*	pointer to next station; NULL if this is the last station on the linked list

TABLE 19: PHAS_INF STRUCTURE

Type	Name	Description of Structure and Data Members
`char`	*ph_id[]*	character string identifying the phase for which information is given within this structure
`Bool`	*prim*	`TRUE` if phase is primary; otherwise, `FALSE`
`float`	*ttime*	travel-time to center of cell (s)
`float`	*ttime_min*	minimum travel-time (s)
`float`	*ttime_max*	maximum travel-time (s)
`float`	*d_ttime_dr*	radial travel-time derivative at cell center (s/deg.)
`float`	*d_ttime_dz*	vertical travel-time derivative at cell center
`float`	*delcell*	cell width in slowness vector space computed from minimum and maximum slowness and azimuthal aperture (width of cell as seen from station)
`Phas_Inf*`	*next*	pointer to next phase; NULL if last

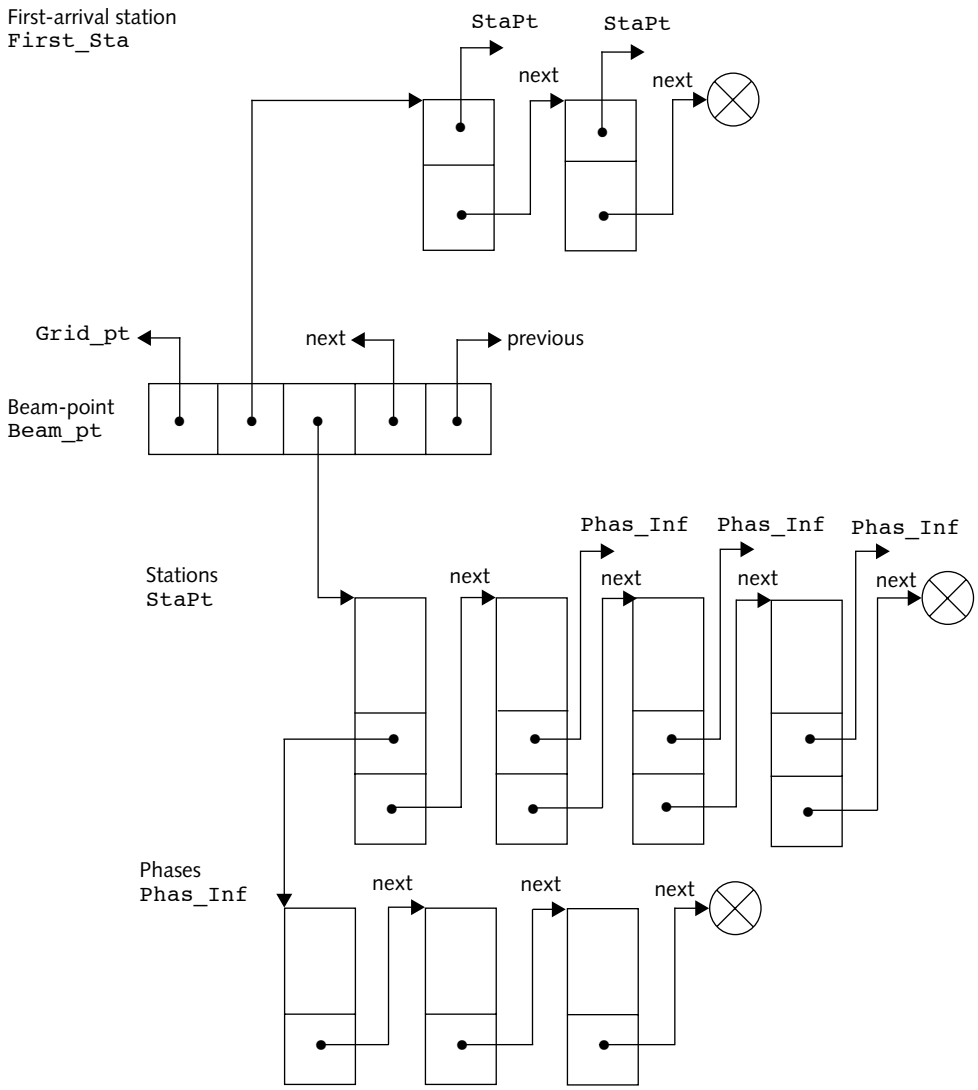


FIGURE 17. RELATIONSHIPS AMONG GACONS DATA STRUCTURES

Error States

The *Build Grid Files* process is not a critical component of the operational system in terms of reliability because it is used only occasionally to build the files containing the knowledge base used by *GAassoc* and *GAconflict*. No failure of this component has been observed, and defensive programming is used to capture any error message and inform the user of the failure condition on the standard output. The most likely source of failure is an erroneous parameter setting such as setting the permissions for a specific directory to deny write privileges. Error messages are logged to standard output and standard error.

DATABASE DESCRIPTION

GAcons use of the database is limited to reading station and network information contained in the **site**, **siteaux**, and **affiliation** tables. These data are accessed using the standard GDI library.

Database Design

GAcons uses the database to obtain the static geographical information for the stations in the network. The entity-relationship diagram of the schema is shown in Figure 4 on page 19. The network (or networks) for which the grid file is computed is selected using the **affiliation** table.

Database Schema

Table 20 shows the usage of database tables by *GAcons*. For each table used, the third column shows the purpose for reading or writing each attribute.

TABLE 20: GACONS DATABASE USAGE

Table	Action	Usage
site	reads	• *sta*, *ondate*, and *offdate* for record identification • *lat*, *lon*, and *elev* for location information
siteaux	reads	• *nois*, *noissd*, *rely*, and *snthrsh* for detection probability
affiliation	reads	• *sta* for record identification • *net* for identifying auxiliary network, or station elements for creating station intervals

Chapter 7: Detailed Design of GAgrid

This chapter describes the detailed design of *GAgrid* and includes the following topics:

- Data Flow Model
- Processing Units
- Database Description

Chapter 7: Detailed Design of GAgrid

DATA FLOW MODEL

GAgrid is a GUI program that allows visualization of one of the two grid files (the Propagation Knowledge Base grid file) built by the *GAcons* program. The information in the grid file is used by *GAassoc* to form trial event hypotheses in the initial phase of the automatic association process. The grid file is an essential component of the automatic association process.

The binary grid file contains of the following information:

- geographic location information for the grid points
- a list of stations in the seismic, hydroacoustic, and infrasonic networks
- the travel-time and slowness information for a list of seismic, hydroacoustic, or infrasonic phases for the paths between each station and each grid cell.

GAgrid provides a graphical display of the grid cells and stations superimposed on a global map with coastlines and political boundaries. The grid content can be examined as alphanumeric or graphical displays.

Figure 18 shows the data flow for *GAgrid*, which consists of two processes, one that reads and parses the grid file and one that displays information at the user's request.

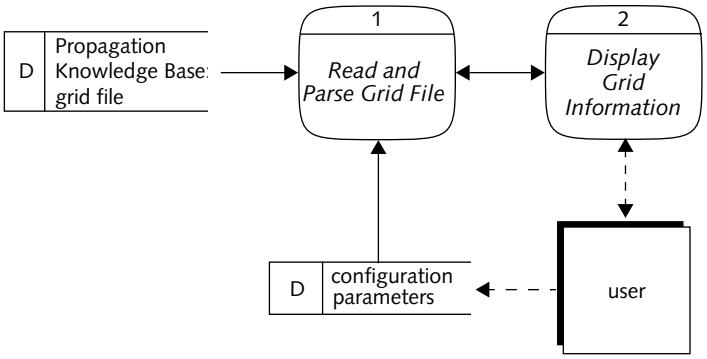


FIGURE 18. GAGRID DATA FLOW

PROCESSING UNITS

GAgrid consists of the following processes:

- *Read and Parse Grid File*
- *Display Grid Information*

The following paragraphs describe the design of these processes.

Read and Parse Grid File

The purpose of the *Read and Parse Grid File* process of *GAgrid* is to provide the interface with the input grid file. At initialization time, the entire content of the grid file is loaded into memory, and the content of the memory is then accessed as needed by the interactive user.

Input/Processing/Output

The main input to the *Read and Parse Grid File* process is the GA grid file containing the propagation information used by *GAassoc* to form the automatic events. The configuration parameter file is the other input; it contains the path and the name of the grid file.

Control

The *Read and Parse Grid File* process is automatically called when *GAgrid* is started. The grid file is read immediately upon initiation of *GAgrid*, and its contents are loaded into memory.

Error handling

As a non-operational software unit, *GAgrid* does not have the same level of robustness as other GA programs; however, errors are trapped and reported in either a terminal or a popup window.

Display Grid Information

Input/Processing/Output

The *Display Grid Information* process uses the information stored in memory by the *Read and Parse Grid File* process of *GAgrid* as input. The processing is driven by the interactive user and consists of graphical and alphanumerical displays. The GA Software User Manual [IDC6.5.12] describes the layout of these graphical and alphanumerical displays in detail. The *libWc*, *libdraw*, and *libXmp* libraries perform the display functions. The alphanumerical displays are displayed using the *libXbae* library.

Control

The user controls the functions that display the data contained within the Propagation Knowledge Base grid file through the use of a pointer device and a set of interactive menus containing buttons with specific purposes. A detailed description of the menus and the functions attached to each of them is provided in [IDC6.5.12].

Error handling

As a non-operational software unit, *GAgrid* does not have the same level of robustness that other GA programs have. However, errors are trapped and reported in either a terminal or a popup window.

DATABASE DESCRIPTION

GAgrid does not use any database table to perform its functions.

References

The following sources supplement are referenced in document:

[Bac93] Bache, T. C., Bratt, S. R., Swanger, H., Beall, G., and Dashiell, F. K., "Knowledge-Based Interpretation of Seismic Data in the Intelligent Monitoring System," *Bulletin of the Seismological Society of America,* Volume 83, pp. 1507–1526, 1993.

[Bra88] Bratt, S. R., and Bache, T. C., "Locating Events with a Sparse Network of Regional Arrays," *Bulletin of the Seismological Society of America,* Volume 78, pp. 780–798, 1988.

[DOD94a] Department of Defense, "Software Design Description," *Military Standard Software Development and Documentation*, MIL-STD-498, 1994.

[Gan79] Gane, C., and Sarson, T., *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.

[IDC5.1.1Rev2] Science Applications International Corporation, Veridian Pacific-Sierra Research, *Database Schema, Revision 2*, SAIC-00/3057, PSR-00/TN2830, 2000.

[IDC5.1.3Rev0.1] Science Applications International Corporation, Veridian Pacific-Sierra Research, Inc., *Configuration of PIDC Databases,* SAIC-01/3022, PSR-99/TN1114, 2001.

[IDC5.2.1] Science Applications International Corporation, *IDC Processing of Seismic, Hydroacoustic, and Infrasonic Data*, SAIC-99/3023, 1999.

References

[IDC6.2.1] Science Applications International Corporation, *System Operation and Maintenance,* SAIC-99/3031, 1999.

[IDC6.5.12] Science Applications International Corporation, *Global Association (GA) Subsystem Software User Manual,* SAIC-01/3003, 2001.

[Kat98] Katz, C. N., Brown, D. J., Gault, A. K., LeBras, R., and Wang, J., P*IDC 6.0: Implementation of Infrasonic Processing in PIDC*, CCB-PRO-98/11, 1998.

[Ken91a] Kennett, B., *IASPEI 1991 Seismological Tables*, Research School of Earth Sciences, Australian National University, 1991.

[LeB96] Le Bras, R., *User Manual for the Global Association Subsystem*, Science Applications International Corporation, SAIC-96/1128, 1996.

[LeB97] Le Bras, R., Sereno, T., Laney, H., Wahl, D., Jenkins, R., Brown, R., Willemann, H., Freese, H., and Renner, B., *Fusion of Seismic and Hydroacoustic Data*, CCB-PRO-97/12, 1997.

[Leo93] Leonard, S., Automatic global event association and location estimation using a knowledge based approach to generalized beamforming, Proceedings of the 15th Annual PL/ARPA Seismic Research Symposium, PL-TR-93-2160.

[Rin89] Ringdal, F., and Kværna, T., "A Multi-channel Processing Approach to Real-time Network Detection, Phase Association, and Threshold Monitoring," *Bulletin of the Seismological Society of America*, Volume 79, pp. 780–798, 1989.

[Tay92] Taylor, D. and S. Leonard, Generalized beamforming for automatic associatin, Proceedings of the 14th Annual PL/ARPA Sefismic Research Symposium, PL-TR-92-2210, 422-428, 1992.

Glossary

A

amplitude

Zero-to-peak height of a waveform in nanometers.

arrival

Signal that has been associated to an event. First, the Global Association (GA) software associates the signal to an event. Later during interactive processing, many arrivals are confirmed and improved by visual inspection.

arrival-quality test

GA test of an event's quality based on the value of the slowness uncertainty and the distance between the event and station for each defining arrival.

arrival tag

Record in the **ga_tag** table that characterizes an arrival to support logic in *GAassoc* and *GAconflict*, for instance as `REQUESTED` for auxiliary seismic arrivals.

associate

Assign an arrival to an S/H/I event.

associated phase

Phase that is associated with an S/H/I event.

association set

Set of arrivals associated with an event hypothesis or confirmed event in GA.

attribute

(1) Database column. (2) Characteristic of an item; specifically, a quantitative measure of a S/H/I arrival such as azimuth, slowness, period, or amplitude.

azimuth

Direction, in degrees clockwise with respect to North, from a station to an event.

B

b value

Slope of the line fit to a plot of seismic magnitude versus cumulative number of events, usually computed for a finite geographic area.

beam point

Data structure used in GA that contains the geographical description of a grid point and the propagation characteristics from all stations in a network to that grid point.

bulletin

Chronological listing of event origins spanning an interval of time. Often, the specification of each origin or event is accompanied by the event's arrivals and sometimes with the event's waveforms.

C

CMR

Center for Monitoring Research.

Comprehensive Nuclear-Test-Ban Treaty Organization

Treaty User group that consists of the Conference of States Parties, the Executive Council, and the Technical Secretariat.

Computer Software Component

Functionally or logically distinct part of a computer software configuration item, typically an aggregate of two or more software units.

Computer Software Configuration Item

Aggregation of software that is designated for configuration management and treated as a single entity in the configuration management process.

confirmed event

An event that passed the minimum weighted count threshold test, an arrival quality test, and a probability of detection test in GA.

conflict resolution

GA process by which arrivals, initially associated to multiple events, are disassociated from all but one of the events.

corroborating arrival

Arrival that is added to an event seeded by the driver arrival that helps to corroborate the preliminary event hypothesis in GA.

COTS

Commercial-Off-the-Shelf; terminology that designates products such as hardware or software that can be acquired from existing inventory and used without modification.

CSC

Computer Software Component.

CSCI

Computer Software Configuration Item.

CTBT

Comprehensive Nuclear-Test-Ban Treaty (the Treaty).

CTBTO

Comprehensive Nuclear-Test-Ban Treaty Organization.

D

DACS

Distributed Application Control System. This software supports inter-application message passing and process management.

defining

Arrival attribute, such as arrival time, azimuth, or slowness, which is used in calculating the event's location or magnitude.

defining arrival

Arrival whose attributes (time, azimuth, and/or slowness) are used to compute an event location.

defining phase

Associated phase for which features are used in the estimation of the location and origin time of an S/H/I event.

deg.

Degrees (as a distance).

detection

Probable signal that has been automatically detected by the *Detection and Feature Extraction* (*DFX*) software.

DFX

Detection and Feature Extraction. *DFX* is a programming environment that executes applications written in Scheme (known as *DFX* applications).

driver arrival

Arrival that is used as an initial seed to build an automatic event in GA. This arrival was detected at one of the stations close to the grid cell being evaluated.

E

entity-relationship (E-R) diagram

Diagram that depicts a set of entities and the logical relationships among them.

event

Unique source of seismic, hydroacoustic, or infrasonic wave energy that is limited in both time and space.

event characterization

IDC process of characterizing events by features of signals recorded at one or more stations.

event hypothesis

Association set that is not yet located or confirmed by the Location and Confirmation process in GA.

G

GA

Global Association application. GA associates S/H/I phases to events.

GB

Gigabyte. A measure of computer memory or disk space that is equal to 1,024 megabytes.

GDI

Generic Database Interface.

grid

Set of points used by GA covering either a region of the Earth or the whole Earth and including the interior where deep seismicity occurs. Information about

propagation to a network of stations is computed by *GAcons* for a grid and stored in a binary file.

grid cell

Volume within the Earth around a grid point in the GA grid that is characterized by the grid point location, a radius, and a depth range around that grid point.

grid point

Location (latitude, longitude, and depth) on the grid used by GA to perform its exhaustive association set search.

GSETT-3

Group of Scientific Experts Third Technical Test.

GUI

Graphical User Interface.

H

hydroacoustic

Pertaining to sound in the ocean.

I

ID

Identification; identifier.

IDC

International Data Centre.

IMS

International Monitoring System.

infrasonic (infrasound)

Pertaining to low-frequency (sub-audible) sound in the atmosphere.

instance

Running computer program. An individual program may have multiple instances on one or more host computers.

I/O

Input/Output.

K

km

Kilometer.

knowledge base

Propagation-characteristic data stored in a binary file generated by *GAcons* and used by *GAassoc*. This file is also referred to as the Propagation Knowledge Base file.

L

large event extractor

Processing module in *GAassoc* that extracts large events (with many defining phases) from the initial list of event hypotheses.

linked list

List of similar data structures related to one another through the use of pointers. A linked list can be uni-directional (the pointer is always to the next element in

the list) or bi-directional (there are pointers to both the previous and next elements in the list).

M

magnitude

Empirical measure of the size of an event (usually made on a log scale).

MB

Megabyte. 1,024 kilobytes.

m_b

Magnitude of a seismic body wave.

M_L

Magnitude based on waves measured near the source.

N

NDC

National Data Center.

network

Spatially distributed collection of seismic, hydroacoustic, or infrasonic stations for which the station spacing is much larger than a wavelength.

network processing

Processing that uses the results of Station Processing from a network of stations to define and locate events.

nondefining phase

Associated phase for which features are not used in estimating the location and origin time of an S/H/I event.

NULL

Empty, zero.

O

ORACLE

Vendor of the database management system used at the PIDC and IDC.

origin

Hypothesized time and location of a seismic, hydroacoustic, or infrasonic event. Any event may have many origins. Characteristics such as magnitudes and error estimates may be associated with an origin.

P

parameter

User-specified token that controls some aspect of an application (for example, database name, threshold value). Most parameters are specified using [*token* = *value*] strings, for example, `dbname=mydata/base@oracle`.

parameter (par) file

ASCII file containing values for parameters of a program. Par files are used to replace command line arguments. The files are formatted as a list of [*token* = *value*] strings.

phase

Arrival that is identified based on its path through the earth.

phase name

Name assigned to a seismic, hydroacoustic or infrasonic arrival associated with a travel path.

PIDC

Prototype International Data Centre.

pipeline

1) Flow of data at the IDC from the receipt of communications to the final automated processed data before analyst review. 2) Sequence of IDC processes controlled by the DACS that either produce a specific product (such as a Standard Event LIst) or perform a general task (such as station processing).

post-location processing

Software that computes various magnitude estimates and selects data to be retrieved from auxiliary stations.

primary phase

First arriving phase recorded at a S/H/I station.

probability of detection

Probability estimate that an arrival from a given event will be detected at a station given the location and magnitude of the event, the average noise level and its standard deviation at the station, and the signal-to-noise detection threshold.

process

Function or set of functions in an application that perform a task.

processing unit

Software component of a larger entity such as a program.

R

radionuclide

Pertaining to the technology for detecting radioactive debris from nuclear reactions.

RDBMS

Relational Database Management System.

REB

Reviewed Event Bulletin; the bulletin formed of all S/H/I events that have passed analyst inspection and quality assurance review.

rollback

Process of returning a database to its original state before processing began.

S

s

Second(s) (time).

SAIC

Science Applications International Corporation.

schema

Database structure description.

seismic

Pertaining to elastic waves traveling through the earth.

SEL1

Standard Event List 1; S/H/I bulletin created by total automatic analysis of continuous timeseries data. Typically, the list runs one hour behind real time.

SEL2

Standard Event List 2; S/H/I bulletin created by totally automatic analysis of both continuous data and segments of data specifically down-loaded from stations of the auxiliary seismic network. Typically, the list runs five hours behind real time.

SEL3

Standard Event List 3; S/H/I bulletin created by totally automatic analysis of both continuous data and segments of data specifically down-loaded from stations of the auxiliary seismic network. Typically, the list runs 12 hours behind real time.

S/H/I

Seismic, hydroacoustic, and infrasonic.

site

Location of a sensor in a station.

slowness

Inverse of velocity, in seconds/degree; a large slowness has a low velocity.

snr

Signal-to-noise ratio.

split event

Event that has been incorrectly formed by GA as several events that associate subsets of the arrivals from the actual event.

sta

Station.

STA/LTA

Short-term average/long-term average ratio.

StaPro

Station Processing application for S/H/I data.

States Parties

Treaty user group who will operate their own or cooperative facilities, which may be NDCs.

station

Collection of one or more monitoring instruments. Stations can have either one sensor location (for example, BGCA) or a spatially distributed array of sensors (for example, ASAR).

station processing

Processing based on data from a single station.

structure

Software construct that collects one or more variables, possibly of different types, together under a single name for convenient handling.

T

Treaty

Comprehensive Nuclear-Test-Ban Treaty (CTBT).

Tuxpad

DACS client that provides a graphical user interface for common Tuxedo administrative services.

U

UNIX

Trade name of the operating system used by the Sun workstations.

W

weighted count

Measure of an event computed as the sum of coefficients for certain arrival attributes such as arrival time, azimuth, and slowness. It is used to define the minimum size of an event hypothesis to be retained by automatic or interactive processing.

wfdisc

Waveform description record or table.

Index

A

affiliation 17, 19
    use by *GA_DBI* 80
    use by *GAassoc* 54
    use by *GAconflict* 73
    use by *GAcons* 94
amplitude 16, 19
    use by *GAassoc* 55
    use by *GAconflict* 73
apma 16, 19
    use by *GAassoc* 55
    use by *GAconflict* 73
arrival 16, 19
    use by *GA_DBI* 80
    use by *GAassoc* 54
    use by *GAconflict* 73
`Arrival_Inf` structure 42
assoc 17, 19
    use by *GAconflict* 74
assoc_ga_temp
    use by *GAassoc* 55
    use by *GAconflict* 74
assoc_temp_ga 18
Automatic Processing CSCI 2

B

`Beam_pt` structure 90
Beam Point record structure 87

C

complementary documents iii
`Cor_Sta` structure 41
CTBT 10

D

DACS 14
database schema
    overview 16
data flow symbols v
data structure conventions vii
design
    conceptual 10
    functional 20
    GA_DBI **75**
    GAassoc **25**
    GAconflict **57**
    GAcons **81**
    GAgrid **95**
    interface 22
`Dr_list` structure 49
`Driver` structure 39

E

entity-relationship
    diagram 19
    symbols vi
ESAL 6
event 17, 19
    use by *GAconflict* 74
event criteria 69

F

file system 15
`First_Sta` structure 90
First-station record 87
functional design 20

G

GA_DBI 11, 13, 18, 22, **75**
database design 79
data flow 76
processing units 77
Tag Auxiliary Arrivals 77
Tag Hydroacoustic Arrivals 78
ga_tag 18, 19
use by *GA_DBI* 80
use by *GAassoc* 55
use by *GAconflict* 73
GAassoc 11, 18, 21, **25**
Access Knowledge Base 32
Associate Arrivals 35
database design 53
data flow model 26
Eliminate Redundant Events 47
Extract Arrival List 30
Extract Large Events 44
Locate and Confirm Preliminary Event Hypotheses 49
processing units 28
Read Command-line Parameters 31
Resolve Conflicts 51
Restrict Phase List 34
Write Event Hypotheses to Database 52
GAconflict 11, 13, 18, 22, **57**
Check Consistency 69
database design 71
data flow model 58
Extract Arrival List 63
Locate and Confirm Preliminary Event Hypotheses 65
Predict Defining Phases 66
Predict Non-defining Phases 68
processing units 61
Read Command-line Parameters 64
Read Event Information 64
Resolve Conflicts 68
Write Event Hypotheses to Database 71
GAcons 11, 18, 21, **81**
Build Grid Files 85
Build Static Grid 83
database design 93
data flow 82
processing units 83
GAgrid 11, 18, 21, **95**
data flow 96
Display Grid Information 98
processing units 97
Read and Parse Grid File 97
GA roles 4
global libraries 14
`Grid_pt` structure 84
Grid Point record 87
GSETT-3 6

H

hardware requirements 7
hydro_arr_group 17, 19
use by *GAassoc* 55
use by *GAconflict* 74
hydro_assoc 17, 19
use by *GAassoc* 55
use by *GAconflict* 73

I

input
database tables 11
interface
IDC systems 22

operator 23
IPC 14

M

man pages iii

N

netmag 17, 19
use by *GAconflict* 74
Network Processing CSC 2

O

operator interface 23
origerr 17, 19
use by *GAconflict* 74
origerr_ga_temp
use by *GAassoc* 55
use by *GAconflict* 74
origerr_temp_ga 18
origin 17, 19
use by *GAconflict* 74
origin_ga_temp
use by *GAassoc* 55
use by *GAconflict* 74
origin_temp_ga 18
output
database tables 11

P

`Phas_Inf` structure 91
Phase record 88
pointer conventions vii
`Pred_triplet` structure 67
processing flow 12
Propagation Knowledge Base grid file 11, 18, 21
structure 86

R

REB 10
reliability 15
requirements
hardware 7
software 7

S

site 17, 19
use by *GAassoc* 54
use by *GAconflict* 73
use by *GAcons* 94
siteaux 17, 19
use by *GAassoc* 54
use by *GAconflict* 73
use by *GAcons* 94
software requirements 7
`Sta_ar` structure 42
stamag 17, 19
use by *GAconflict* 74
`StaPt` structure 90
Static grid file 11, 18, 21
Station record 88
structure data content 37

T

typographical conventions vi