# The Complete Developer's Guide to the MAGS System

Aaron W. Hsu
arcfide@sacrideo.us

Karissa McKelvey
krmckelv@indiana.edu

Joshua Cox
joshcox@indiana.edu

1st Edition

# Chapter 1
# Introduction to MAGS

## 1.1 Welcome to MAGS

## 1.2 Design Goals

# Chapter 2
## Architectural Design Overview

# Chapter 3

# Application Server

## 3.1 Introduction to the Application Server

The MAGS Application Server is the glue that binds the various components that make MAGS into a single, integrated application. It is meant to be the single unifying program, and as such, it is generally expected that most modifications of MAGS will not need to touch the core Application Server proper. At the heart, the application server controls access to the database server. It manages student rosters, grades, and runs the autograder on submissions it receives. The application server is not a front-end, but middleware between front-end clients like Guido and the database and autograder reports. The server itself is written in Dyalog APL, which provides for a rapid, concise code base. It is neutral to the database server used and serves its clients through SOAP exchanges. Should the need for additional services arise, they are easily added using the simple and easy to use SAWS framework.

The server is deployed as a single, standalone workspace. This is much like a Smalltalk image for those familiar with that language. This workspace contains a latent expression that automatically launches the server. When hacking the server, one usually opens the `websrv.dws` workspace in a Dyalog session and manages the code files (also called scripts) from there. The script files in the `websrv/` directory—they have the `.dyalog` extension— correspond to namespaces in the workspace. In modern version of Dyalog, changes made to namespace scripts in the workspace will automatically be reflected in the script files after the user confirms a prompt.

Initializing, creating, managing, and removing the database is also done through the `websrv.dws` workspace. Commands for setting server options, as well as starting and stopping the server also exist in the workspace. As you can see, nearly everything is managed through the application server. Each of these commands is documented in more detail further on, but this discussion gives you a few hints to get your bearings. To begin delving deeper, we will start at the ends, the database schema and the Web Service interface, and then move to the middle, describing the methods and variables of the various namespaces in detail.

## 3.2 Database Schema and Approach

The application server uses a fairly simple database schema. Our relational model is believed to be in BC normal form and should have no redundancy of data storage. Moreover, we explicitely do not permit `null` data in any attribute. Most relations have only a simple natural key and we avoid any artifical keys. Each table may be an entity, a relation, or mixed. Mixed tables are entities with certain bijective relations merged into them. All fields contain atomic data. We make heavy use of foreign and normal key constraints, but few other constraints are necessary for our model.

This chapter documents the database schema in detail and should be considered the authoritative reference. Inconsistencies between this document and the application server code should be treated as serious bugs. The following is a short summary of each table's purpose; each heading in this section focuses on detailing one of these tables.

| Entities | Relations | Mixed |
|---|---|---|
| Instructor | Validates | Submitter |
| Groups | Teaches | Submission |
| Assignment | HasType | |
| Problem | HasDeadline | |
| Category | BelongsTo | |
| Comment | Contains | |
| Deadline | Collaborated | |
| Validator | CommentsOn | |

**Instructor Entity**. This entity holds the set of all non-students, whether teachers, assistents, or just graders. A member of this entity is allowed to grade assignment submissions based on what groups they may "teach." Thus, not all instructors may grade all submissions.

| Field Name | Type | Foreign Key |
|---|---|---|
| `networkid` | Fixed String | — |
| `firstname` | Variable String | — |
| `lastname` | Variable String | — |

Natural Key: `networkid`
Participates in these relations: Teaches

**Group Entity**. This entity currently records the names of various groups, which may correspond to either internal groups for a class or to class sections or any other division. It serves as the divider by which submitters are grouped. Assignments are also assigned to specific groups.

| Field Name | Type | Foreign Key |
|---|---|---|
| `name` | Variable String | — |

Natural Key: `name`
Participates in these relations: Submitter.member, Teaches, and BelongsTo

**Assignment Entity**. An assignment is a named collection of problems. Problem Set might be another good name for it. Assignments are given to groups of students and have auto-graders associated with them in the form of validators and problem auto-graders. When a submitter submits a submission, it is submitted for a particular assignment. Assignments may be reused or used for particular groups (that is, used for more than one group). Assignments can be categorized.

| Field Name | Type | Foreign Key |
|---|---|---|
| `name` | Variable String | — |

Natural Key: `name`
Participates in these relations: Submission.submittedfor, Validates, HasType, HasDeadline, BelongsTo, Contains, Collaborated, CommentsOn

**Problem Entity**. Problems are the central entity in MAGS. They represent the unit of grading and most everything else is based on the concept of grading a problem. Autograders are applied to individual problems. They are composed of a name, a description, a test suite, and a solution.

| Field Name | Type | Foreign Key |
|---|---|---|
| `name` | Variable String | — |
| `description` | Variable String | — |
| `testsuite` | Variable String | — |
| `solution` | Variable String | — |
| `grader` | Variable String | — |
| `grader_params` | Variable String | — |

Natural Key: `name`
Participates in these relations: Contains, CommentsOn

**Submitter Entity/Relation**. This mixed entity/relation captures the concept of a student or submitter of solutions. Each submitter belongs to a group. Currently, a submitter may be a member of only one group, but this may change in the future.

| Field Name | Type | Foreign Key |
|---|---|---|
| `networkid` | Fixed String | — |
| `firstname` | Variable String | — |
| `lastname` | Variable String | — |
| `member` | — | `Group.name` |

Natural Key: `networkid`
Participates in these relations: Submission.owner, Collaborated, CommentsOn

**Submission Entity/Relation**. A submission is a proposed solution to a given assignment submitted by a submitter. It has a submission date associated with it and may be marked as an appeal, meaning it is a resubmission of a previous submission. The autograder report is attached to the submission directly because submissions are intended to be graded immediately on submission and should have only a single report.

| Field Name | Type | Foreign Key |
|---|---|---|
| `submissionid` | Serial (Integer) | — |
| `owner` | — | `Submitter.networkid` |
| `submittedfor` | — | `Assignment.name` |
| `date` | Date | — |
| `isappeal` | Boolean | — |
| `code` | Variable String | — |
| `report` | XML | — |

Natural Key: `owner`, `submittedfor`, `date`
Participates in these relations: Collaborated, CommentsOn

**Category Entity**. Categories are simple organizers for assignments. They are largely informational.

| Field Name | Type | Foreign Key |
|---|---|---|
| `name` | Variable String | — |

Natural Key: `name`
Participates in these relations: HasType

**Deadline Entity**. A deadline is some date with a type tag that can be used to track certain features of a submission, such as whether it was submitted on time. Assignments may have many such deadlines. While the natural key is the `type` and `date`, we use a `deadlineid` field to make matching in foreign keys doable.

| Field Name | Type | Foreign Key |
|---|---|---|
| `deadlineid` | Serial (Integer) | — |
| `type` | Variable String | — |
| `date` | Date | — |

Natural Key: `type`, `date`
Participates in these relations: HasDeadline

**Validator Entity**. A validator is a program which will be run to verify that a submission is acceptable before running the autograder on it. A failed validator results in rejecting a submission. Unlike autograders, validators are associated with assignments.

| Field Name | Type | Foreign Key |
|---|---|---|
| `name` | Variable String | — |
| `command` | Variable String | — |

Natural Key: `name`
Participates in these relations: Validates

**Comment Entity**. A comment is just some text. We have an entity for it because Guido has a concept of reusing comments.

| Field Name | Type | Foreign Key |
|---|---|---|
| `text` | Variable String | — |

Natural Key: `text`
Participates in these relations: CommentsOn

**Validates Relation**. Indicates that a given validator should be used to validate submissions for a particular assignment. It also prescribes any additional parameters for use when calling the validator.

| Field Name | Type | Foreign Key |
|---|---|---|
| `validator` | — | `Validator.name` |
| `assignment` | — | `Assignment.name` |
| `params` | Variable String | — |

Natural Key: `validator`, `assignment`, `params`

**Teaches Relation**. Binds or associates a member of Instructors as assigned to or grading a group.

| Field Name | Type | Foreign Key |
|---|---|---|
| `teacher` | — | `Instructor.networkid` |
| `group` | — | `Group.name` |

Natural Key: `teacher`, `group`

**HasType Relation**. Associates an assignment with a category. Assignments may belong to multiple categories.

| Field Name | Type | Foreign Key |
|---|---|---|
| `assignment` | — | `Assignment.name` |
| `category` | — | `Category.name` |

Natural Key: `assignment`, `category`

**HasDeadline Relation**. Assigns or gives an assignment a deadline, where deadlines have types and dates.

| Field Name | Type | Foreign Key |
|---|---|---|
| `assignment` | — | `Assignment.name` |
| `deadline` | — | `Deadline.deadlineid` |

Natural Key: `assignment`, `type`, `date`

**BelongsTo Relation**. Associates an assignment to a group. Assignments may belong to more than one group.

| Field Name | Type | Foreign Key |
|---|---|---|
| `assignment` | — | `Assignment.name` |
| `group` | — | `Group.name` |

Natural Key: `assignment`, `group`

**Contains Relation**. Assigns a problems to an assignment with a given problem number. Assignments should not reuse a problem, but multiple assignments may use the same problem.

| Field Name | Type | Foreign Key |
|---|---|---|
| `assignment` | — | `Assignment.name` |
| `problem` | — | `Problem.name` |
| `number` | Variable String | — |

Natural Key: `assignment`, `problem`

**Collaborated Relation**. Indicates other students with whom the submitter of a submission collaborated with during the completion of an assignment. While other collaborations may have occurred, only submitter collaborations are currently tracked.

| Field Name | Type | Foreign Key |
|---|---|---|
| `submission` | — | `Submission.submissionid` |
| `collaborator` | — | `Submitter.networkid` |

Natural Key: `submission`, `collaborator`

**CommentsOn Relation**. An instructor comments on a submission by taking a comment and associating it with a start and end position somewhere in the submission. A comment may be used in multiple places in an assignment. Additionally a comment is marked as relating to a specific problem. Please note the unmentioned constraint here, that the problem that is commented on should be one of the problems contained by the assignment for which the submission was submitted. If this is not the case, then who knows what could happen.

| Field Name | Type | Foreign Key |
|---|---|---|
| `comment` | — | `Comment.text` |
| `submission` | — | `Submission.submissionid` |
| `problem` | — | `Problem.name` |
| `start` | Int | — |
| `end` | Int | — |

Natural Key: `comment`, `submission`, `problem`, `start`, `end`

## 3.3 Web Services

## 3.4 Public Interface

## 3.5 Workspace Overview

**Reading the code**.

## 3.6 The Main Namespace

## 3.7 The WS Namespace

# Chapter 4
## Guido Manual Commenting Interface

# Chapter 5
Submission Validation

# Chapter 6
# Interfacing with Autograders or Writing Your Own

The main MAGS application server has no built-in auto-grading support. Auto-grading is very specific to the domain and especially the language being used. To make MAGS able to handle many different environments, we have no specific interface for auto-graders except by standard input and standard output. Auto-graders are invoked as shell commands with specific, standard arguments. The test results should be printed to standard output. The output is in XML format. Most programming languages that may benefit from MAGS have mature XML capabilities so this should not be an imposition. The rest of this chapter focuses on these two interfaces: the command-line format and the XML schema. Once an auto-grader has been designed according to these specifications, it need only be installed in the binary path of the application server before it may be used.

## 6.1 Command-line Interface

The command-line interface described here gives the arguments with which each instance of the auto-grader is called. In particular, each problem in an assignment will spawn one instance of the auto-grader. Each invocation is expected to be referentially transparent, meaning that multiple runs of an auto-grader with the same arguments should produce the same output. One should not rely on hidden state to get reasonable results. The pattern fo arguments looks like this:

```
<grader> : test-suite solution submission [extra ...]
```

Each problem is associated with a `test-suite` and a `solution`, so these will change on a per problem basis. The `submission` is provided as the unique student submission. It is tied to the code field of the student submission database. The `extra` fields are given on a per problem basis to provide more information to the auto-grader if necessary. If an error occurs during grading, this should be reported through the return value of the program according to the standard operating system conventions.

## 6.2 XML Result Schema

Auto-graders return grading results via standard output in an XML format documented here. In particular, grading results are divided into tests that are expected to either pass or fail. The result of a test indicates what the expected result was as well as the actual result of the test. In addition to the pass/fail result of the test and the name of the test, there are an arbitrary number of properties that can store additional information. The actual properties used in an auto-grader are chosen by the specific auto-grader and may not even use the same set of properties for each test in a given run. That is, all properties are optional. Tests may be grouped into test groups, but need not be. Test groups may also nest. All results are contained inside a root `grading-results` node. What follows is a description of the nodes and their attributes.

Node Name: `grading-results`
Attributes: —
Valid Child Nodes: `test-group`, `test-result`
Description: Root node of the auto-grader output.

Node Name: `test-group`
Attributes: `name`
Valid Child Nodes: `test-group`, `test-result`
Description: Groups together a set of testing results.

Node Name: `test-result`
Attributes: `name`, `result`, `expected`
Valid Child Nodes: `test-property`
Description: Gives the result of a single test, its pass or fail status, and the name of the test.

Node Name: `test-property`
Attributes: `name`
Valid Child Nodes: Character data
Description: Gives a single property of a test result. The value of the property is given as data in the property, while the name is given as an attribute.

### 6.3 Handling Errors

If a critical error in grading occurs, the auto-grader is not expected to produce valid result data provided this situation is adequately reported in the return code. In this case, the output thus far will be used in the error but will not be used as report data.

# Chapter 7
Scheme Autograder

# Chapter 8
Java Autograder

# Chapter 9
APL Autograder

# Chapter 10
Python Autograder

# Chapter 11
Quick Reference Material