OSEHRA

Open Source Electronic Health Record Agent

Source KIDS White Paper

26 August 2011

SUBJECT:

OSEHRA technical investigation into "Source KIDS"

1. Purpose

This white paper proposes development of "Source KIDS" in order to represent VistA software in a source tree suitable for use with modern version control tools.

2. Overview

VistA development currently takes place inside MUMPS environments. Developers move changes among VistA instances using KIDS builds. They may not use modern editors or version control tools that work with source files on disk because these tools have no access to the software. We propose development of "Source KIDS" (SKIDS) to overcome this problem and enable a development workflow as illustrated in Figure 1.

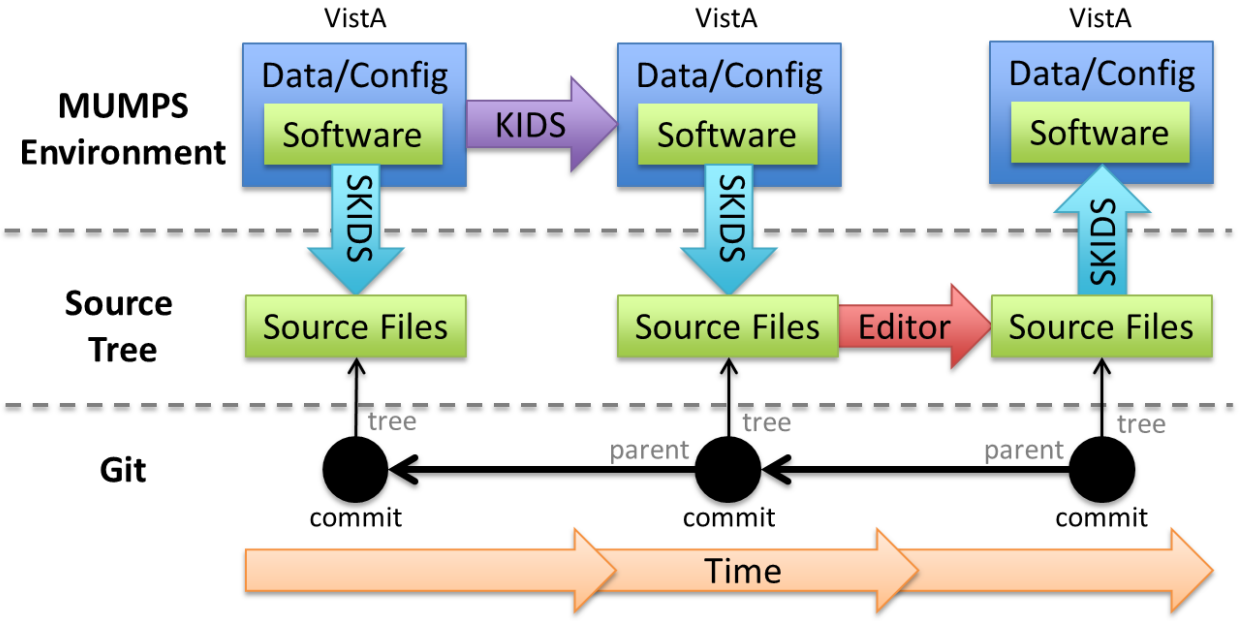

Figure 1: SKIDS Workflow Overview

In this workflow a developer may use SKIDS to export a representation of the VistA software, excluding data and configuration specific to one VistA instance, from a MUMPS environment into source files organized in a traditional filesystem tree and commit it to a version control repository. After applying a KIDS build the developer may export the software again and commit the new version to the repository. One may then inspect the version control history to identify the differences introduced by the KIDS build. Then a developer may edit the source file representation directly on disk and commit a new version to share with collaborators in the version control repository. Finally, a developer may use SKIDS to import the VistA software changes from the source tree back into a MUMPS environment.

3. Description

Medsphere Systems Corporation describes SKIDS on a project page at medsphere.org:

> *"Source KIDS (SKIDS) is an effort to **make VistA compatible with modern distributed revision control tools** such as Bazaar. The general approach is to **leverage KIDS' awareness of internal VistA structures** such as options and bulletins to **represent them as regular source code-like files**, then use Bazaar to manage these files. If the text files are modified externally, they can be re-imported into VistA to effect the changes made to the text files."*
>
> – Jonathan Tai, http://**medsphere.org**/thread/1553

Although this description mentions Bazaar it applies equally well to many version control tools including Git. No further information or updates are available on the page at the time of this writing.

Medsphere's description exactly matches the problem that we must solve in order to achieve the above development workflow. The VistA software within a MUMPS environment consists of two parts. First, a collection of MUMPS routines contain the majority of the implementation. Second, a set of "non-routine software elements" contain reference data for use by the implementation (e.g. remote procedure call definitions). These data are static with respect to a given VistA version, are part of the software, and do not change during normal operation. We must represent both parts as source files in order to use modern version control tools.

VistA routines may be represented simply as ".m" source files. Unfortunately VistA's non-routine software elements have no straightforward representation as source files. A MUMPS environment stores them in globals and does not distinguish globals representing software elements from those holding configuration settings or patient-specific data located within VistA. The KIDS implementation knows how the software elements are structured within MUMPS globals and how to extract them for transportation to another VistA instance in the form of a KIDS build. KIDS' knowledge can help create a source file representation for VistA's structures; hence the name "Source KIDS".

4. Design

SKIDS will leverage KIDS' knowledge to represent VistA's software elements as source files outside of a MUMPS environment. Most data in VistA, including the non-routine software elements, are stored in FileMan files. FileMan provides VistA's main organizational structure. The KIDS implementation knows how to transfer FileMan files from one VistA instance to another in the form of a KIDS build (.KID). Figure 2 shows two of the data export dialogs that select FileMan files for transfer. KIDS also pre-defines about 20 "build components", listed in Figure 3, that collect all the information needed to represent and transfer a logical VistA

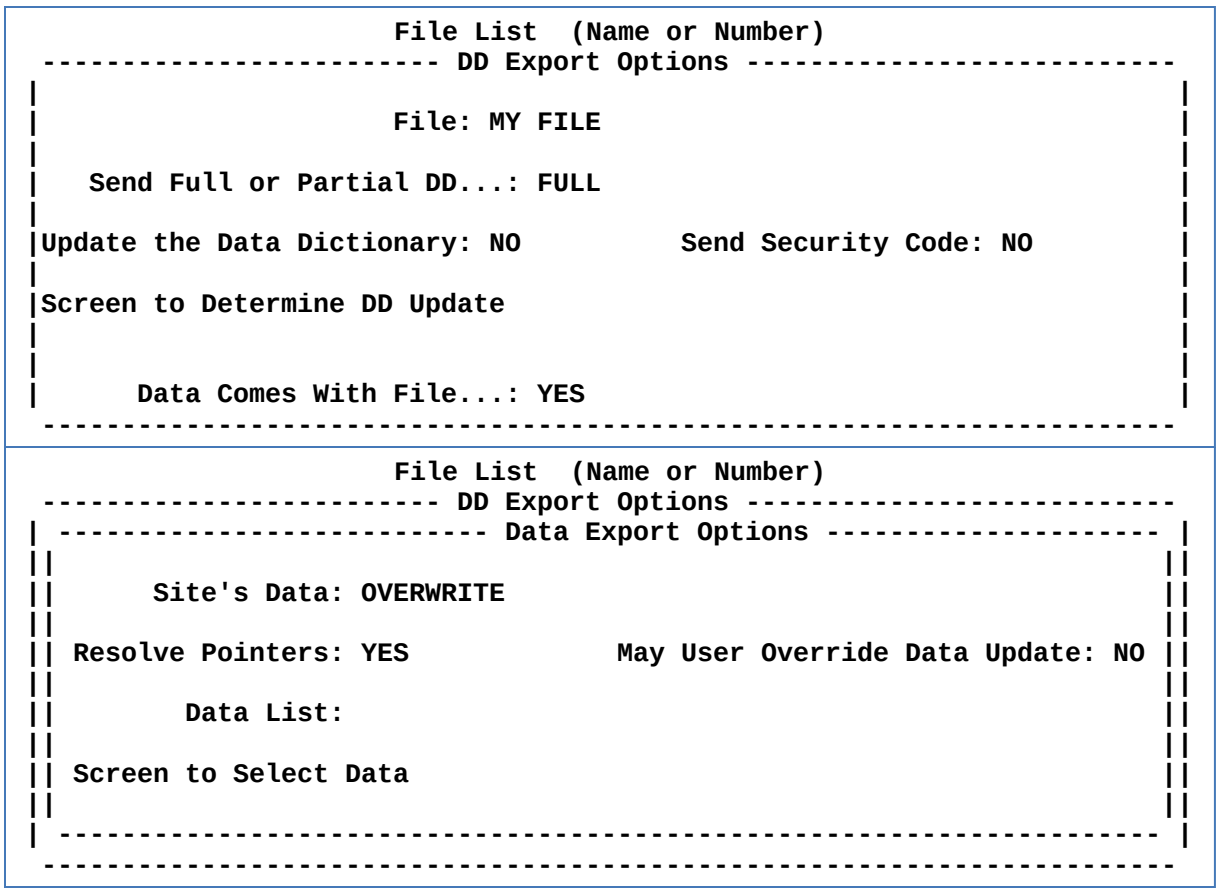

Figure 2: KIDS Data Export Screens

component such as a remote procedure call definition.

FileMan files have a centrally managed unique number, name, and documented purpose. Data within each file are organized into a list of *records*. Each record contains data that logically belong together, such as information about one patient, organized as a set of *fields*. The Data Dictionary defines for each file the fields held in its records. Every field has a name, type, documentation string, and content validation transform. SKIDS can utilize the Data Dictionary to map FileMan file records into a text-based representation in a source file.

```
        Build Components

PRINT TEMPLATE                (0)
SORT TEMPLATE                 (0)
INPUT TEMPLATE                (0)
FORM                          (0)
FUNCTION                      (0)
DIALOG                        (0)
BULLETIN                      (0)
MAIL GROUP                    (0)
HELP FRAME                    (0)
ROUTINE                       (0)
OPTION                        (0)
SECURITY KEY                  (0)
PROTOCOL                      (0)
LIST TEMPLATE                 (0)
HL7 APPLICATION PARAMETE      (0)
HLO APPLICATION REGISTRY      (0)
HL LOGICAL LINK               (0)
PARAMETER DEFINITION          (0)
PARAMETER TEMPLATE            (0)
REMOTE PROCEDURE              (0)
```

Figure 3: KIDS Build Components

FileMan assigns to every record an integer identifier unique within its file, but the identifiers are meaningful only within a single VistA instance. Many FileMan files define a NAME field in each record to identify it with a meaningful string. FileMan maintains cross-references that are used for lookups or other purposes, including a default NAME cross-reference that maps each NAME to a list of records that contain it. These cross-references must not be included in a source file representation because they are meaningful only within a given VistA instance and can be re-computed as necessary. However, the NAME of each record is meaningful and must be preserved.

One may display the definition of any file or the data in any record through interactive FileMan sessions in programmer mode. Figure 4 shows two sessions that display the Data Dictionary's definition of the REMOTE PROCEDURE file and the data contained in its ORQQVI VITALS record, respectively. SKIDS must define a text-based representation of FileMan file records, such as ORQQVI VITALS, for storage in source files. The representation must preserve enough information to reconstruct all fields in the original record. For example, Figure 5 shows a manually constructed representation of the record in JavaScript Object Notation (JSON).

Since each FileMan file may contain thousands of records, and each record holds logically grouped data, one may sensibly map each FileMan file to a source directory named after the file and map each record to a source file in said directory. For example, the content shown in Figure 5 may be stored in a source file at a path such as

`"Files/REMOTE PROCEDURE/ORQQVI VITALS.json"`

to represent the ORQQVI VITALS record in the REMOTE PROCEDURE file.

```
VISTA>D P^DI
VA FileMan 22.0
Select OPTION: DATA DICTIONARY UTILITIES
Select DATA DICTIONARY UTILITY OPTION: LIST FILE ATTRIBUTES
 START WITH WHAT FILE: REMOTE PROCEDURE// REMOTE PROCEDURE
       GO TO WHAT FILE: REMOTE PROCEDURE// REMOTE PROCEDURE
       Select SUB-FILE: <ENTER>
Select LISTING FORMAT: STANDARD// <ENTER>
Start with field: FIRST// <ENTER>
DEVICE: HOME// <ENTER>
```

```
VISTA>D P^DI
VA FileMan 22.0
Select OPTION: INQUIRE TO FILE ENTRIES
OUTPUT FROM WHAT FILE: REMOTE PROCEDURE// REMOTE PROCEDURE
Select REMOTE PROCEDURE NAME: ORQQVI VITALS
     1   ORQQVI VITALS
     2   ORQQVI VITALS FOR DATE RANGE
CHOOSE 1-2: 1  ORQQVI VITALS
ANOTHER ONE: <ENTER>
STANDARD CAPTIONED OUTPUT? Yes// <ENTER>   (Yes)
Include COMPUTED fields:  (N/Y/R/B): NO// <ENTER>
```

Figure 4: Display REMOTE PROCEDURE file definition and ORQQVI VITALS record

5. Representation

Design of a suitable source file representation of FileMan files is a critical component of SKIDS. The JSON representation used in the previous section serves as a proof-of-concept but may or may not be sufficient as a general solution. The design requirements include:

- **Human Readable**: The representation must be meaningful to human readers and editable by programmers. Although a KIDS build (.KID) is text-based, the format is not readable and cannot be safely edited. The JSON example in the previous section meets this requirement.
- **Version Control Friendly**: The representation must be suitable for standard line-wise version control tool operations such as diff, blame, and merge. Furthermore it must be deterministic and reproducible such that exporting an identical record from two VistA instances produces an identical source file representation. This makes version control history reflect the true changes made to the software instead of idiosyncrasies of the export mechanism. Since JSON is whitespace agnostic outside quoted strings it will be suitable only with a standard for whitespace usage.
- **Lossless**: The representation must hold complete FileMan records. A round-trip conversion to a source file and back into a FileMan record must preserve all data. This includes handling of special field types such as POINTER and VARIABLE POINTER. The JSON example in the previous section does not cover such special types.

```
{
 "NAME": "ORQQVI VITALS",
 "TAG": "FASTVIT",
 "ROUTINE": "ORQQVI",
 "RETURN VALUE TYPE": "ARRAY",
 "AVAILABILITY": "RESTRICTED",
 "DESCRIPTION": [
  "Array of patient most recent vitals within start and stop date/times.  If",
  "no start and stop dates are indicated, the most recent are returned.",
  " ",
  "If no start date is passed then the start date is 1 (i.e. before any",
  "dates).",
  " ",
  "If no stop date is passed then the start date is also the stop date and if",
  "there is not start date then 9999999 is used as the stop date."
 ],
 "INPUT PARAMETER": [
  {"INPUT PARAMETER": "PATIENT ID",
   "PARAMETER TYPE": "LITERAL",
   "MAXIMUM DATA LENGTH": "16",
   "REQUIRED": "YES",
   "DESCRIPTION": [
    "Patient id (DFN) from Patient File (#2)."
   ]
  },
  {"INPUT PARAMETER": "START DATE/TIME",
   "PARAMETER TYPE": "LITERAL",
   "MAXIMUM DATA LENGTH": "16",
   "DESCRIPTION": [
    "Start date/time for vital retrieval in Fileman format.",
    "If none is passed then the start date is 1 (i.e. before any dates)."
    ]
  },
  {"INPUT PARAMETER": "STOP DATE/TIME",
   "PARAMETER TYPE": "LITERAL",
   "MAXIMUM DATA LENGTH": "16",
   "DESCRIPTION": [
    "Stop date/time for vital retrieval in Fileman format.",
    "If none is passed then the Start date is also the stop date and if there",
    "is not start date then 9999999 is used as the stop date"
   ]
  }
 ],
 "RETURN PARAMETER DESCRIPTION": [
  "Array of patient most recent vitals within start and stop date/times.",
  "If no start and stop dates are indicated, the most recent are returned.",
  "Vitals are returned in the format:",
  "vital ien^vital type^rate/value^date/time taken"
 ]
}
```


Figure 5: Text-based (JSON) representation of ORQQVI VITALS record of REMOTE PROCEDURE file

- **Minimal**: The representation must not include redundant or extraneous information from the globals FileMan uses internally. Cross-references and internal record identifiers must be excluded.

Additional requirements may be encountered during the design investigation. For example, the Data Dictionary itself must be represented. In addition to FileMan files and records, SKIDS must also support custom representations for the 20 KIDS "build component" types listed in Figure 3. Ideally the component representations will re-use the above representation of FileMan records as a primitive when possible. Those components that contain information beyond a single FileMan record must be represented in a form that groups the information together or otherwise identifies the relationship of its parts.

6. Approach

We propose a three-phase approach to the design, implementation, and use of SKIDS.

1. **Raw Globals**: Initially we will naïvely represent VistA's non-routine software elements using raw exports of the MUMPS globals storing them (in %GO or ZWR format). This does not meet any of the requirements (except perhaps "lossless") but is sufficient for storing VistA software versions and to install a new VistA instance from scratch.
2. **FileMan Files**: After designing a source file representation for FileMan records we will add source files representing VistA's FileMan files and remove the corresponding raw globals. This transformation may be completed incrementally as support for more Data Dictionary field types is added to SKIDS.
3. **Build Components**: Given the FileMan record representation as a primitive, we will design custom formats for specialized KIDS build components. This will make the components available in the source tree in a form suitable for custom component editors.

After completion of these three phases we will inspect any remaining raw globals to determine what specialized representation they need, if any.

7. Build System

KIDS incorporates pre- and post-initialization routines, pre-install questions, and dependency checks. These elements also need to also be presented in a format that conforms to the representation requirements above. The format will serve as a sort of "Makefile" for the SKIDS system that makes it functionally equivalent to KIDS. Since all parts of a KIDS build will be represented as source files additional functions may be possible. Perhaps:

- Automatic generation of KIDS builds for (non-)developers.
- Incorporation of changes from other source trees (both SKIDS-based and KIDS-based).
- Allow for the normal updating of VistA instances including packages, versions, etc.

8. Summary and Conclusions.

The overall goal of the SKIDS system is to create a format that is programmer readable and editable and can be tracked by version control systems containing VistA Routines and "non-routine software elements". The SKIDS system accomplishes this goal by defining a file format and directory structure that leverages the built-in knowledge of the existing KIDS system. The system will be built using phases starting with a raw global export, then adding support for FileMan files, and finally adding support for the twenty build components that KIDS defines. SKIDS also needs a build system that contains the installation instructions that are typically found in a KIDS build such as pre-installation questions, pre- and post-installation routines, and dependency checks.

This document defines SKIDS and the problems that must be solved to make it work. However, "the devil is in the details." Due to the complexity of the system, a team with a variety of background and expertise is required for effectively undertaking this effort. Following best Open Source practices, we will create an open and transparent working group supported by the OSEHRA community enablement infrastructure. We invite participation from members of the VistA community at large. Working together we will bring the power of modern version control tools to VistA development.

OSEHRA Contractor Team