Software Tool House Inc. **Meta-Update**

Samples Guide

Release 6250
Updated: 2025-Jun-01

Preface

Audience

This document is intended for Remedy ARS and/or ServiceNow Administrators and developers.

It is expected that the reader will have knowledge of the Remedy ARS system and be familiar with workflow development. It would behove the reader to be familiar with his ARS server's platform and scripting tools.

Limitation of Liability

Copyrights

Updates

This program and this manual may change from time to time. The latest version is available at our web site: www.softwaretoolhouse.com.

Comments

Your comments are welcome! Please see: www.softwaretoolhouse.com/support and click **Comments**, or email us at support@softwaretoolhouse.com. We look forward to hearing from you!

Software Tool House Inc

Document Library

The following documents are included with Meta-Update.

File	Contents
Meta-Update Installation Guide	Meta-Update.and the Job Console installation guide.
Meta-Update Users Guide.	This is a detailed reference on Meta-Update scripting. It is used by script developers. It covers developing and debugging scripts.
Meta-Update Samples Guide *This document.*	This is a detailed reference on many of the Meta-Update sample scripts. The samples do useful things and this document can be used for learning Meta-Update scripting. Templates for the samples are installed with the Job Console application.
Meta-Update Job Console Users Guide	This is a detailed reference on developing templates and firing jobs using the Job Console.
Trace Daemon Users Guide	The “Trc” version of the binaries communicate with a process called the trace daemon. This is the User Guide for implementing and using this process.
Meta-Update Release Notes	This highlights changes made in this release of Meta-Update.

Organisation

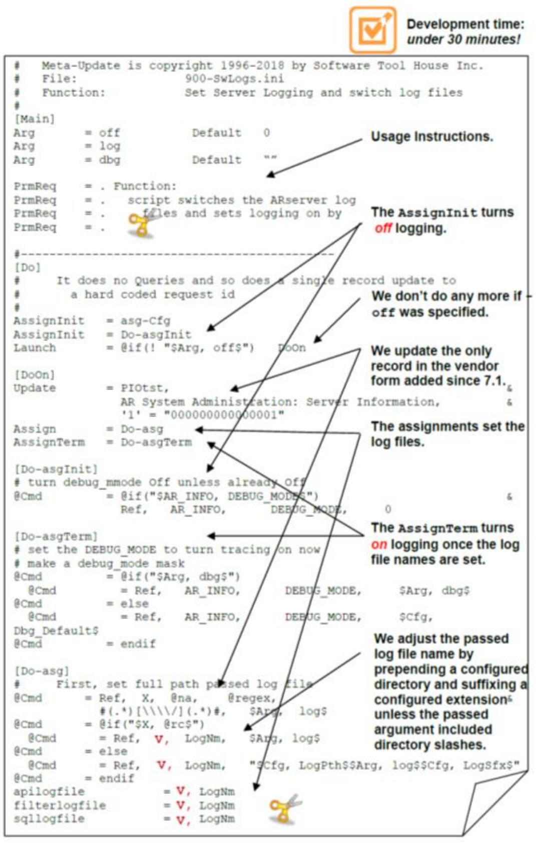
This document outlines the samples included in the samples directory of the extracted distribution.

It is expected that the reader

- has installed Meta-Update on his local workstation, and,
- has generated an **`SthLic.cmd`** or **`SthLic.sh`** file. See Meta-Update Installation Guide if needed.
- has read at least the Concepts section of the Meta-Update User's Guide

This document is split into three sections:

- It starts with a list and short description of scripts in the **`samples`** folders of the Meta-Update distribution.
- It then gives a brief overvoew of each sample.
- Finally, it gives detailed descriptions of the scripts using images of the script with explanations in boxes.



Development time: *under 30 minutes!*

```
#   Meta-Update is copyright 1996-2018 by Software Tool House Inc.
#   File:               900-SwLogs.ini
#   Function:           Set Server Logging and switch log files
#
[Main]
Arg         = off           Default   0
Arg         = log
Arg         = dbg           Default   ""

PrmReq      = . Function:
PrmReq      = .   script switches the ARserver log
PrmReq      = .     files and sets logging on by
PrmReq      = .

#-----------------------------------------------------
[Do]
#     It does no Queries and so does a single record update to
#       a hard coded request id
#
AssignInit   = asg-Cfg
AssignInit   = Do-asgInit
Launch       = @if(! "$Arg, off$")   DoOn

[DoOn]
Update       = PIOtst,                                              &
               AR System Administration: Server Information,        &
               '1' = "000000000000001"
Assign       = Do-asg
AssignTerm   = Do-asgTerm

[Do-asgInit]
# turn debug_mmode Off unless already Off
@Cmd         = @if("$AR_INFO, DEBUG_MODE$")                         &
               Ref,   AR_INFO,      DEBUG_MODE,     0

[Do-asgTerm]
# set the DEBUG_MODE to turn tracing on now
# make a debug_mode mask
@Cmd         = @if("$Arg, dbg$")
  @Cmd         = Ref,   AR_INFO,      DEBUG_MODE,     $Arg, dbg$
@Cmd         = else
  @Cmd         = Ref,   AR_INFO,      DEBUG_MODE,     $Cfg,
Dbg_Default$
@Cmd         = endif

[Do-asg]
#     First, set full path passed log file
@Cmd      = Ref,  X,  @na,    @regex,
            #(.*)[\\\\/](.*)#,   $Arg,  log$
@Cmd      = @if("$X, @rc$")
  @Cmd      = Ref,  V,  LogNm,   $Arg, log$
@Cmd      = else
  @Cmd      = Ref,  V,  LogNm,   "$Cfg, LogPth$$Arg, log$$Cfg, LogSfx$"
@Cmd      = endif
apilogfile          = V, LogNm
filterlogfile       = V, LogNm
sqllogfile          = V, LogNm
```

Usage Instructions.

The **`AssignInit`** turns *off* logging.

We don't do any more if **`off`** was specified.

We update the only record in the vendor form added since 7.1.

The assignments set the log files.

The **`AssignTerm`** turns *on* logging once the log file names are set.

We adjust the passed log file name by prepending a configured directory and suffixing a configured extension unless the passed argument included directory slashes.

Figure 1 Detail Descriptions of a Sample Script

Document Conventions

Typefaces and conventions and icons are used in this document to add specific meaning as follows:

Icon & Type Conventions	Meaning
	Windows specific. Does not apply to Linux.
	Linux specific. Does not apply to Windows.
bmc	Applies to BMC Remedy ARS server sessions. Cannot be used for, or does not refer to ServiceNow sessions..
now	Applies to ServiceNow sessions. Cannot be used for, or does not refer to BMC Remedy ARS server sesions.
	Caution. Failure to follow recommended actions may cause data loss.
`Courier Bold`	**`Courier Bold`** indicates a command you can enter. For example: **`set SthApiRetry=90-92 0 60 93 0 30`** **`export SthApiRetry=90-92 0 60 93 0 30`**

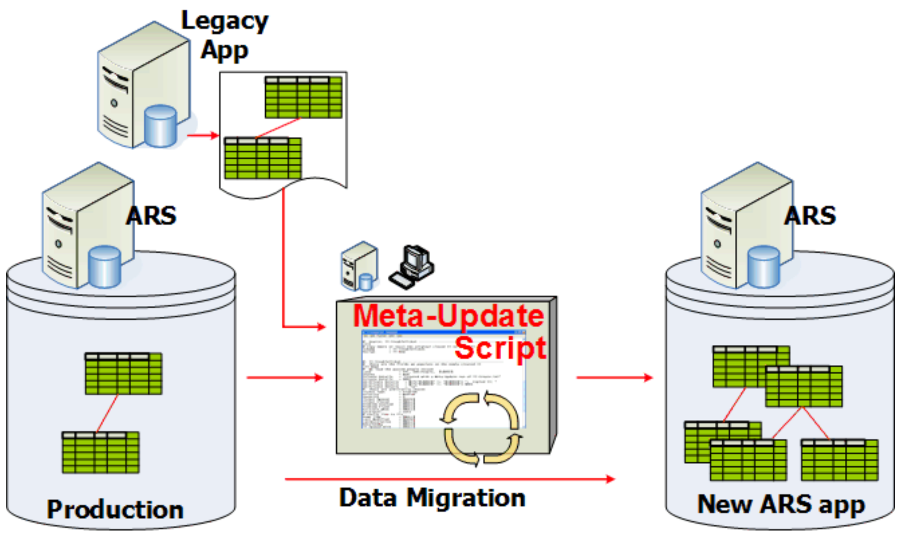
Table of Contents

Introduction

Introduction

Thank you for selecting Meta-Update. With Meta-Update, creating repeatable imports, migrations and batch operations on your ARS data is a snap.

Don't bother with the API! Meta-Update provides a quick, robust, reliable, auditable method of harnessing the power of the API without ***any*** programming at all.



Data Challenges

- Ever had trouble setting up an ARS data migration?
  - From one server version to another?
  - From one release of ITSM to another?
  - From ITSM 6 or 5 or 4 to ITSM 9.1?
  - From a bespoke ticketing and asset system to another different bespoke application, to an ITSM implementation?

- Ever had trouble importing data into an ARS application?
  - From a series of CSV files representing complex data trees?
  - From CSV files that Excel or the import tool can't handle: containing embedded new-lines, and field values with embedded, undoubled quotes?
  - From CSV files where the query to determine the update record is complex?
  - From CSV files where the target update form changes for each row in the data?
  - From fixed length transactional files / records?

- Ever had trouble getting data transformations right?
  - Assigning the right Status values based upon a different set of incoming values and more complex conditions?
  - Selecting the fields to be updated based upon incoming transaction data, queried data, read data?
  - Setting the values based upon incoming transaction data, queried data, read data?
  - Assigning values to reserved fields like Create Date, and Submitter.

- Ever wanted to adjust, correct, merge, and change the ARS data that you have?
  - Ever needed to combine two clients' foundation data records?
  - Ever wanted to rename or split up support groups?
  - Ever needed to automate the importing of foundation data into the ITSM suite?

- Ever had trouble creating an ARS API program?
  - Ever wasted time talking with a non-ARS programmer?
  - Waited when making assignment or form logic changes for the programming development cycle before seeing the results?

Meta-Update: A New Way to Use The API

With Meta-Update, these types of problems are handled quickly, with ease and confidence!

There is no need for an API programmer or ***any*** programmer at all.

The ARS Administrator / Developer scripts complex functions in the language he already knows in minutes. He fine tunes mappings and assignments and gets his feedback immediately. His runs are fully logged allowing complete resolution and recovery.

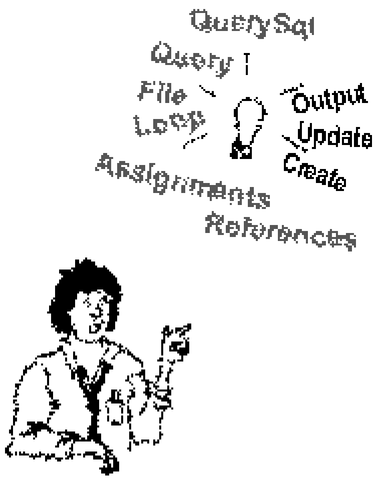
Development efforts for any migration or file import requirements are reduced to at least 1/10th.

That’s an order of magnitude savings on the initial development effort compounded by fewer resources required to maintain or enhance scripts from the deployment on.

Compound that development savings with the confidence you get by using Meta-Update:

- The performance is that of the API run on the server or client.
- Jobs complete with “Log and Continue” error processing.
- Errors produce complete resolution and retry information logs.
- Jobs can be broken up in batches and run simultaneously on one or more machines.
- Core fields can be easily assigned on both primary and secondary forms.
- CSV files that fail on the import tool can be handled easily.
- Transactional files can be handled.
- Dates, times, users, status history can be set to any desired value.
- Diary fields’ entries can be looped through creating records in other forms.
- All ARS permissions and workflow is respected.

Concepts


QuerySql
Query
File
Loop
Output
Update
Create
Assignments
References

Running Meta-Update

In this section, we will cover:

- Setting up the run time environment
- BMC Remedy API versions
- Meta-Update program versions
- Using the license keys
- Environment variables
- The Meta-Update command line usage
- Meta-Update output and return values
- Meta-Update Tracing

Run Time Environment

Meta-Update runs in a Windows "Command Prompt" or UNIX shell. It is a simple process that can be fired by workflow, batch files, shell scripts, even Meta-Update scripts.

Scripts and files developed and referenced may be interchanged freely between Window and UNIX.

Meta-Update scripts can be run

- By users of the Job Console application
- manually in a shell or command prompt
- in a filter with the $PROCESS$ actions
- through a batch file or shell or Perl script
- through an OS scheduler like **cron** or **at**.

The runtime environment is the same for workflow, script, and manual operation.

The Meta-Update "`bin`" directory contains all required Meta-Update binaries or executable programs, shared objects and dlls.

The Meta-Update `bin` directory should be on the path.

On Windows, the Meta-Update "`bin`" directory can be set in the PATH= environment variable with:

```
Set PATH=D:\Apps\Sth\Meta-Update-5.56\;%PATH%
```

The program operates in a Command Prompt, or "DOS Box", or as a fired process. Local trace files are written in the current working directory by default.

On Solaris or Linux, the Meta-Update "`bin`" directoriy needs to be in the PATH= and LD_LIBRARY_PATH= environment variables.

```
export PATH=/Apps/Sth/Meta-Update-5.77/bin/:$PATH
 export LD_LIBRARY_PATH=/Apps/Sth/Meta-Update-
5.77/bin/:$LD_LIBRARY_PATH
```

The program operates under any of the available shells or as a spawned or background process. Local trace files are written in the current working directory when not specified.

BMC Remedy API Versions

Meta-Update is generally compiled against the most current BMC supplied version of the BMC Remedy API. The Meta-Update distribution includes all BMC supplied dlls that are required.

The Meta-Update API version does not need to match the version of the servers that Meta-Update establishes with. Meta-Update can establish multiple connections to different Remedy servers of different releases.

Software Tool House always recommends that the highest API version is used no matter what your server version is.

ServiceNow API & System Properties

Meta-Update uses the current ServiceNow REST API. It uses libcurl to setup connections to any ServiceNow instances.

The Meta-Update distribution includes all dlls that are required. See https://github.com/curl/curl for libcurl information.

System Properties Changes recomended

ServiceNow, ***by default***, will return ***all*** records for queries with *invalid* qualifications.

Some Meta-Update scriptrs accept query terms on the command line. A typo in a field name will lead to all records satisfying the query and being processed by the script.

A specific System Property can be added that prevents this behaviour and returns zero records for invalid query qualifications.

This is a very dangerous property to be missing by default. Any errors in any query qualification text, such as mis-typed field names, will cause ***all*** records of the table to be returned.

For example, a script to delete records based on a query argument can accidentally delete all records if passed a mistyped field name.

The System Property to prevent this action and instead return zero records when a query qualification is in error, is named: **`glide.invalid_query.returns_no_rows`**.

It must be set to true and the record created if missing from the **`sys_properties`** table.

Meta-Update, by default will check that this is set, and quit if not.

There is a command line argument that controls this behaviour and can be used to set this value on each ServiceNow instance the script references.   Each instance needs to be set for Meta-Update to run against it, by default. It needs to be set once on each instance.

This argument can be specified on any run for each ServiceNow instance.

`-snQryChk`	**`quit	set	ignore`**
`quit`	is the default and causes Meta-Update to end with an error and do no updates at all.
`set`	will add the system property that returns no records on invalid queries – a much safer option, and
`ignore`	will check for this system property and only give a warning – a very dangerous operation.

There is also a sample script that can be used to create this record. Note that you must use the above argument with **`ignore`** to run it..

```
SthMupd.exe   samples\700-SN\900-SvrProp-set.ini Do
              -key  glide.invalid_query.returns_no_rows
              -type boolean
              -val  true

SthMupd.exe   -SnQryChk set      anyscript Do -anyarg 1

SthMupd.exe   -snQryChk ignore   anyscript Do -anyarg 1
```

You can also create this record manually using the ServiceNow interface. Simply create a new record in **`sys_properties`** using name:
`glide.invalid_query.returns_no_rows` and value **`true`**.

Program Versions

There are two versions of Meta-Update and bundled utilities with different names. One is used for local tracing and the other includes tracing through a trace server. These programs have different names. They are the same name in all operating systems:

`SthMupd.exe`	Local trace version
`SthMupdTrc.exe`	Trace server version

Logging is controlled by the Meta-Update **`-d`** switch in the same way across versions. See The Command Line below for more information on the **`-d`** switch.

The local trace version always appends to a file named **`SthMupd.log`** in the current directory unless the trace file is named with the **`-d`** switch.

With the Trace server version, traces are sent to the trace server. The trace server is administered to record selected levels of traces and discard other levels. The trace server version, needs both the **`-d`** switch, and the trace daemon set correctly for debugging traces to be captured

The trace server must be running on the same machine as Meta-Update. Communication to the trace server is with the standard message queue facility under Unix or with Named Pipes under Windows.

If the Trace Server version of Meta-Update is run, and the trace server is not started, Meta-Update will act as though the local trace version was run. That is: a file named **`SthMupd.log`** in the current directory is appended to unless the trace file is named with the **`-d`** switch.

More information can be found on the Trace facility in Server Tracing below, and the document, The Common Trace facility.

The License Key

You need a license key to run Meta-Update. Please see Licensing below for more information on licensing Meta-Update and obtaining License Keys.

You can tell Meta-Update the license key in one of these ways:

- Use **`SthLic.cmd`** or **`SthLic.sh`** for convenience

- Code it on the command line with the **`-lic`** argument
- Code it in the script itself with **`[Main] License=`**
- Set an environment variable with it as done with **`SthLic.cmd`** and in the samples

The environment variable to be set is **`SthMupdLic`**. In the script, you can specify **`License=`** in the **`[Main]`** section.

A utility is used to generate an **`SthLic.cmd`** Windows batch file, or **`SthLic.sh`** bash shell script. This is a convenient way to set licensing, server and authentication parameters. It also allows ARS User passwords to be encrypted. See SthLicUpd Maintenance Utility below.

Environment Variables

Both Meta-Update and the BMC Remedy API can be affected by using Environment Variables[1]. This section defines the Meta-Update environment variables and the values and behaviours associated with them.

BMC Remedy documentation is the accurate source for documentation on the BMC API environment variables. We summarize them here because they affect Meta-Update behaviour.

Meta-Update environment variables are fully defined below:

Environment Variable	Description
`SthScriptPath`	A path-like environment variable for finding Meta-Update scripts and files.
`SthApiRetry`	Allows Meta-Update to retry API operations on any BMC Remedy API errors or during server outages.
`SthMupdLic`	Specifies the Meta-Update license key for the main server.

BMC Remedy API environment variables are specified in the BMC provided documentation. The usage of these variables may be changed at any time. This list is included for convenience and because it affects and overrides Meta-Update behaviours. Validate all usage of these variables with your Remedy documentation.

Environment Variable	Description
`ARAPILOGGING`	Generates two files in the current working directory of the running Meta-Update process. Conflicts will occur when multiple Meta-Update processes with this environment variable are run.
`ARTCPPORT`	Sets **all connections** TCP Port to the servers. Overrides the Meta-Update **`Port=`** keyword which can be different for different servers.
`ARRPC`	Specifies a private RPC port for all server connections.

Script Path Environment Variable

Scripts may be specified on the command line or may be found by searching an **`SthScriptPath`** environment variable.

SthScriptPath is set the same way as PATH according to the OS that Meta-Update is running on.

On Windows, one could set the script path like this:

[1] "**Environment variables** are a set of dynamic named values that can affect the way running processes will behave on a computer." - Wikipedia

```
                    set
      SthScriptPath=E:\Projects\ITSM\Scripts;D:\Apps\STH\samples\;
```

On LINUX, one could set the path like so:

```
                    export
      SthScriptPath=/Projects/ITSM/Scripts/:/Apps/STH/samples:
```

Note the difference in the path and directory separators.

Subdirectories in the paths are not searched. However if the script passed to the command line contains a relative path, that relative path will be checked against the **SthScriptPath** and the first matching file will be opened.

API Retry Environment Variable

A Meta-Update job normally returns any errors received from the ARS server during any of its API calls and cancels the single record it was processing. It would then continue with the next record.

It is useful to protect the Meta-Update run from a server timeout, crash, or restart. Meta-Update can retry some API calls to the server based on configurable ARERR codes, a maximum number of retries, and a delay between retries.

The environment variable SthApiRetry= may be used to specify these retry settings.

Without this environment variable, all API calls that fail cause an error in Meta-Update that can result in a record being lost, not found, or the Meta-Update job terminating before processing all records of a query.

The **SthApiRetry=** string is either a single or multiple sets of three numbers:

```
                    start_ARERR_number [ - stop_ARERR_number ]
      Retries Delay
```

start_ARERR_number [- stop_ARERR_number]	Single or ranges of ARERR numbers can be specified.
Retries	A **Retry** count of 0 means infinite number of retries.
Delay	The **Delay** is in seconds. A **Delay** of 0 means no delay.

The following example illustrates its use to protect against servers crashes and servers that have timed out.

```
      set SthApiRetry=90-92 0 60 93 0 30
```

```
                    export SthApiRetry=90-92 0 60 93 0 30
```

These examples retry API calls resulting in error 90, 91, 92, 93, retrying an infinite number of times, with a 30 second delay on ARERR 93 (timeout due to busy server) and a 60 second delay for ARERR 90, 91, 92.

Note that for Query timeouts (94), retries will generally not resolve the problem. Instead use the **`TimeOutLong=`** keyword of the **`[Main]`** section.

fs

License Environment variable

```
SthMupdLic    =      license-key
```

If this environment variable is defined, the license check is made against the value associated.

This is primarily used on the server and also in high performance situations.

```
        AnyVar         =      Value
```

Any environment variable may be used in a Meta-Update script. All defined environment variables are referenced by the reserved tag, **`ENV`**. The field name is the environment variable name.

Environment variables, like all other field names are case sensitive.

```
        Loop           =      String, Pth, ”;”, $ENV, PATH$
```

The above example loops for every directory in the PATH environment variable.

As another example, the environment variable, **`ArsGlobals = 5`**, could be used to load a site-specific set of values and keys to other records.

```
        LoadQ  =      Tag, Schema, ‘1’ = $ENV, ArsGlobals$
```

The Command Line

A Meta-Update command at a minimum specifies the Meta-Update script and the starting section within that script.

That script may require arguments and Meta-Update accepts built-in switches – for example to run the debugger or increase logging detail.

Scripts can have named arguments that can be coded in any order before or after the script and section.

```
>>> SthMupd.exe 090-SvrAdmin\220-SwLogs.ini Do -log tst1
I terminating successfully in 2 sec.
```

By convention, in this document and in our samples, script arguments are specified after the script file and section name.

```
>>> SthMupd.exe 090-SvrAdmin\221-SwLogs.ini Do
E Line 28 - required argument -log not on command line; no
default specified
E . Function:
E . This is a Meta-Update script that switches the ARserver log
files
E .
E . Usage
E .   SthMupd   221-SwLogs Do   -log  xxx
E .     where       xxx         is a log file name without a
path
E .                               and without the .log
E .                             The path and ".log" are
configurable
E .                               in the script
E . Examples
E .   SthMupd   221-SwLogs Do   -log  my
E .             will set all log files to:
"/apps/bmc/ARSystem/db/my.log"
E .
E terminating unsuccessfully in 2 sec.
```

Meta-Update has a set of switches that may be specified on the command line. Each script can also define a set of arguments that may be set on the command line or defaulted to a value.

Entering the Meta-Update command with no arguments yields usage help. Entering the Meta-Update command with the single –help switches yields more detailed help.

```
SthMupd.exe
SthMupd.exe  -help   |   more
```

Switches

Entering the Meta-Update command with no arguments or the single **`-help`** switch yields usage help.

```
SthMupd.exe
SthMupd.exe    |    more
```

Logging	
`-d`	Specifies logging. By itself, all specified full debugging logs to the default log file with no ARS Server logging and no Debug2 logging.
`--d`	As above but includes Debug2 logging and ignores any Trace assignment commands in the script.
`-q`	Inhibits echoing of specific logs to the console but does not affect the logging file.
`-v`	Verbose. Equivalent to –d:qas All field structures, queries, and data values are logged.
Development switches	
`-e`	Single error mode. Stops execution of the script when the first error is encountered.
`-g`	Debugging more. Enters the Meta-Update debugger.
Server switches	
	Note that servers and authentication may be specified on the command line, in the script, or default to the environment variables set by the **`SthLic.cmd`** batch file. Defaults for the Main server when not coded on the command line or in the script are the environment variables:
`-ServerType xxx`	Specifies or overrides the main server type: ARS or SN **`ARS`** Specified that the server is a Remedy server (default) **`SN`** Specifies a ServiceNow instance URL
`-server xxx`	Specified the main ARS server connect address or ServiceNow instance URL. May be an IP or machine name. May also point to a specific server of a load-balanced server group or the load balancer address.

`ArsTyp`	**`ARS`** or **`SN`** for Remedy and ServiceNow respectively
`ArsSvrAdmin`	The server name or IP.
`ArsPort`	The server port. Use of the port mapper is the default and can be specified with zero.
`ArsUsr`	The ARS or ServiceNow user that Meta-Update will be running under. Note that this user generally has administrator rights.
`ArsPwd`	The encrypted or plain text password of the ARS or ServiceNow user that Meta-Update will be running under.

`-port xxx`	Specified the main ARS server's port number. Zero is the default and indicates that the port mapper is used. Not used for ServiceNow
`-user xxx`	Specified the main ARS server's or Admin's ServiceNow login user that Meta-Update will be running under. Note that this user is generally an administrator.
`-password xxx`	Specified the ARS or ServiceNow user's password. May be plain text or encrypted with **`SthLicUpd.cmd`**.
Other switches	
`-help`	Summary usage instructions.

Usage Help Text

```
Meta-Update    Version 5.80 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2018 by Software Tool House Inc.
               www.softwaretoolhouse.com

Function:
  SthMupd runs a Meta-Update script at the specified section
              against a BMC Remedy Server and/or ServiceNow instance.
  See: http://www.softwaretoolhouse.com for the User's Guide and Licensing.

Synopsis:
  SthMupd [ switches ]    script-file    section    [ script-arguments ]

    The script-file and section must follow each other.

    Switches and arguments have the form:    -switch    [ value ]
    The script can include named arguments which are specified by using the script's
         argument name as the switch followed by the value for that argument.
    The script should explain its usage when run with no switch arguments.

    script-file    is the Meta-Update script to run; may be found in the path-like
                     Environment Variable: SthScriptPath
    section        a section to process in the script file  ("Do" for samples)

    switches for logging;  Warning: Produces large output and slows throughput.
      -d           Full tracing into SthMupd.log with no '2' or ARS server tracing
     --d           Full tracing like -d, plus: '2' and ignores script Trace commands
     -d:x,y,f      Tracing:  x   specifies tracing levels: qsad2flp
                             y   ARS client tracing flags: fsap
                             f   is the tracing file name (local or Caution: global)
      -q,-quiet    Quiet:         inhibit all output to stdout (not log!)
      -v           Verbose:       same as -d:qsa
    switches for script development:
      -g           Debug Mode:    enter script debugger; "help" for commands.
      -e           single Error:  terminate job on first error (for script dev/test)

    switches for specifying [Main] server   Note that servers must be licensed.
                                            Set defaults with SthLic.cmd
     -ServerType  ARS | SN           Server Type              default: ENV, ArsTyp
     -server      server             Server                   default: ENV, ArsSvrAdmin
                                                                       ENV, ArsSvr
     -user        user               server's ARS user        default: ENV, ArsUsr
     -password    Enc:xxx            ARS User's password      default: ENV, ArsPwd
     -port        port               server's ARS Port or 0   default: ENV, ArsPort
     -locale      locale[.charset]   server's locale setting  default: ENV, ArsLocale

    other switches
      -snQryChk   set | quit | ignore     check ServiceNow servers' sys_properties'
                                          glide.invalid_query.returns_no_rows setting
                                                   default: quit

130719.518 i terminating successfully in 0 sec.
```

In the local trace version, the –d switch causes a high level of tracing. This data is appended to a file that will grow if not deleted occasionally. Without the **–d**, the file will still be continually added to, but at a much reduced volume. Only Error, and other informational messages will be written. See Tracing below for more information.

In the Trace Server version, the **–d** switch causes a lot of message traffic between Meta-Update and the Trace daemon. The trace files are cycled through and do not grow beyond the limits specified in the trace configuration. See ***Tracing*** for more information.

The **–q** switch indicates quiet operation. No messages will be echoed to the stdout or stderr files at all. This includes all Error and Info messages as well as the copyright notice. These messages will still appear in the logs.

The **–n** switch indicates a null operation. No database writes are performed but all queries and loads are processed. The assignments are also processed and the updating data is printed to the console. This may be useful when you are developing a new script file. Note that with complex scripts, because no database writes are performed, references needed may not exist.

The **–e** switch indicates a “single error” operation. The first error that occurs will stop the run. Use this when developing new scripts.

Normally, a file or query is processed and sections that are launched may succeed or fail. If a launched section fails, then the remaining records in the file or query continue to be processed. Using the **–e** switch changes that behaviour so that the job ends when the first error happens.

When developing scripts, this allows the developer to sort out each section in sequence quickly.

The **script-file** parameter is the name of the file containing the Meta-Update controls and the target record assignments. It must exist and read access must be permitted for the user running Meta-Update.

The **ArSvr**, **ArUsr**, **ArPwd**, and, **ArPort** parameters will override similar parameters in the Main section of the script file. If they are not coded in the assignment file, they are required on the command line.

If **ArSvr** is coded, the **ArUsr**, **ArPwd**, are also required, and **ArPort** is required if the listed server does not use Port Mapper. The command line arguments cause the equivalent script file keywords to be overridden and ignored.

There is an encryption utility provided to encrypt ArsUsr passwords. Generally, one would set these in the file and let the operating system’s file security prevent unauthorised access to that file. This and encryption would keep the ARS User and password secure. In the script, these may be set to environment variables or other references.

Script arguments are specified as a minus followed by the named argument. Any value following that is considered the value of that argument. The script may specify defaults (including NULL) and then that argument is not required. See [Main] Section and Arg – Program Arguments.

Wrap long values in quotes according to your shell as needed.

Program Return Values

The program returns a zero upon successful completion. If ***any*** errors occur, the program returns 1. This value may be used in scripts to decide a course of action.

Errors and important informational messages are reported the trace file. They are also echoed to stderr, generally the console.

stderr may be redirected. On UNIX and Windows, the syntax is the same:

```
      SthMupd.exe    . . .      2>>errors.txt
```
Or
```
      SthMupd.exe    . . .       2>errors.txt
```

The first command appends between runs. The second creates a new file each time.

This file may be examined with any ASCII editor such as Notepad, Word, vi… The format of the trace messages are explained further in Tracing below.

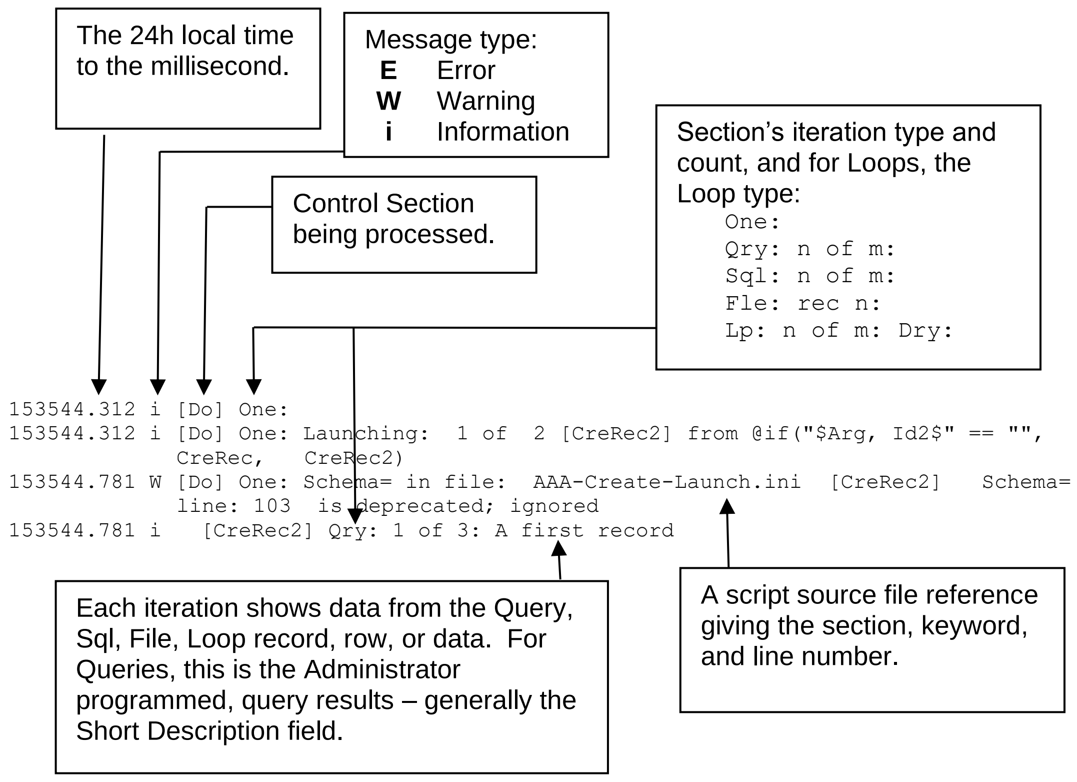
Note that error messages are also always written to stderr, which is generally the console window. If redirected as in the above example command invocations, Errors and Warnings may be grep'd or find'd from this file. See Tracing below for more information.

Program Output

Unless the –q switch is used, Informational, Warning, and Error messages are echoed to the console. These messages tell you what section is working on what record and lists outputs to ARS tables. These messages are also captured in the trace logs.

An example:

```
E:\Dta\_wrk\ > SthMupd.exe AAA-Create-Launch.ini Do -p 426 429

Meta-Update    Version 5.56 (x64) for ARS lib 8.1.2
           (c) Copyright 1996-2015 by Software Tool House Inc.
               www.softwaretoolhouse.com
153544.312 i [Do] One:
153544.312 i [Do] One: Launching:  1 of  2 [CreRec2] from @if("$Arg, Id2$" == "",
             CreRec,   CreRec2)
153544.781 W [Do] One: Schema= in file:  AAA-Create-Launch.ini  [CreRec2]   Schema=
             line: 103  is deprecated; ignored
153544.781 i   [CreRec2] Qry: 1 of 3: A first record
153544.890 i   [CreRec2] Qry: 1 of 3: Merged schema: _Test, Id: 000000000004474 OldId=
153544.921 i   [CreRec2] Qry: 2 of 3: and now, only seconds lat
153544.968 i   [CreRec2] Qry: 2 of 3: Merged schema: _Test, Id: 000000000004475 OldId=
153544.968 i   [CreRec2] Qry: 3 of 3: A second entry made a few
153545.031 i   [CreRec2] Qry: 3 of 3: Merged schema: _Test, Id: 000000000004476 OldId=
153545.031 i   [CreRec2] Qry: eof 3 record OK; 0 records with errors; total: 3.
153545.031 i [Do] One: Launching:  2 of  2 [CopyRec2] from @if("$Arg, Id2$" == "",
             CopyRec,  CopyRec2)
153545.031 W [Do] One: Update0= in file:  AAA-Create-Launch.ini  [CopyRec2]   Update0=
             line: 98  is deprecated.  Use AssignNew=
153545.031 i   [CopyRec2] Qry: 1 of 3: A first record
153545.125 i   [CopyRec2] Qry: 1 of 3: Merged schema: _Test, Id: 000000000004477
             OldId=
153545.125 i   [CopyRec2] Qry: 2 of 3: and now, only seconds lat
153545.187 i   [CopyRec2] Qry: 2 of 3: Merged schema: _Test, Id: 000000000004478
             OldId=
153545.187 i   [CopyRec2] Qry: 3 of 3: A second entry made a few
153545.234 i   [CopyRec2] Qry: eof 3 record OK; 0 records with errors; total: 3.
153545.234 i [Do] One: 1 record OK; 0 records with errors; total: 1.
153545.234 i Statistics:
153545.234 i         Sections:                    3
153545.234 i         Maximum section depth:       2
153545.234 i         Assignment Sections:         6
153545.234 i         Singleton Sections:          1   errors:      0
153545.234 i         Queries:                     2
153545.234 i         Query records:               6   errors:      0
153545.234 i         Output Schemas:              0
153545.250 i         Output Schema records:       6   created
153545.250 i         Output Schema records:       0   updated  (with 0 skipped)
153545.250 i         Outputs OK:                  6
153545.250 i         Outputs Errors:              0
153545.250 i         Outputs Aborts:              0
153545.250 i         Input   Errors:              0
153545.250 i terminating successfully in 1 sec.

E:\Dta\_wrk\ >
```



The 24h local time to the millisecond.

Message type:
E Error
W Warning
i Information

Control Section being processed.

Section's iteration type and count, and for Loops, the Loop type:

```
One:
Qry: n of m:
Sql: n of m:
Fle: rec n:
Lp: n of m: Dry:
```

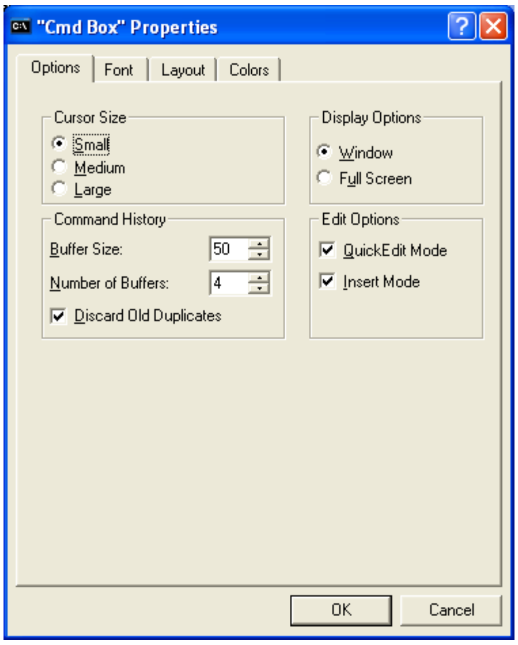
```
153544.312 i [Do] One:
153544.312 i [Do] One: Launching:  1 of  2 [CreRec2] from @if("$Arg, Id2$" == "",
             CreRec,   CreRec2)
153544.781 W [Do] One: Schema= in file:  AAA-Create-Launch.ini  [CreRec2]   Schema=
             line: 103  is deprecated; ignored
153544.781 i   [CreRec2] Qry: 1 of 3: A first record
```

Each iteration shows data from the Query, Sql, File, Loop record, row, or data. For Queries, this is the Administrator programmed, query results – generally the Short Description field.

A script source file reference giving the section, keyword, and line number.

Ideal Command Prompt Properties

Software Tool House recommends that for the convenience of the Meta-Update script developer, the Command Prompt have a wider and deeper buffer and that Quick Edit mode be set. This applies to the UNIX shell as well.


"Cmd Box" Properties

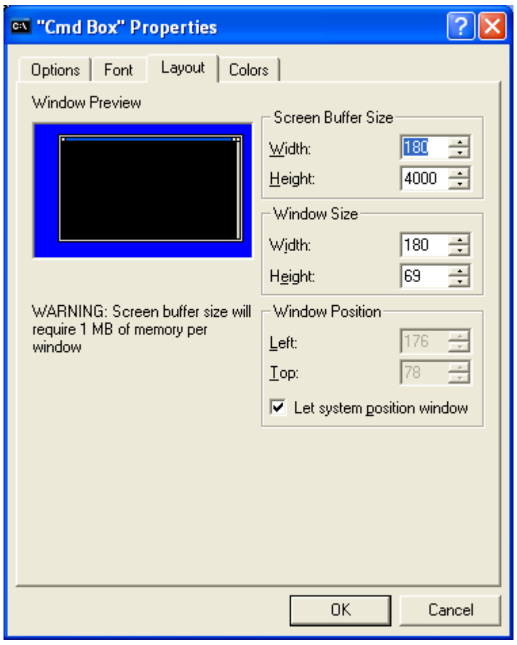
Options | Font | Layout | Colors

Cursor Size
- [✓ Small]
- Medium
- Large

Display Options
- [✓ Window]
- Full Screen

Command History
- Buffer Size: 50
- Number of Buffers: 4
- ☑ Discard Old Duplicates

Edit Options
- ☑ QuickEdit Mode
- ☑ Insert Mode


"Cmd Box" Properties

Options | Font | Layout | Colors

Window Preview

Screen Buffer Size
- Width: 180
- Height: 4000

Window Size
- Width: 180
- Height: 69

Window Position
- Left: 176
- Top: 78
- ☑ Let system position window

WARNING: Screen buffer size will require 1 MB of memory per window

On Windows, click the Command Prompt Icon on the Title Bar, select Properties and ensure that QuickEdit Mode is on and then increase your Buffer Size Width and Height.

In addition, we highly recommend that “Cygwin” be installed, and Meta-Update script developers become familiar with it. There are numerous utilities that are especially useful for handling large log files.

“Cygwin” provides open source LINUX-like utilities and shells for Windows. It is available at www.cygwin.com

Tracing

Tracing can be controlled through the use of the –d switch. When a –d is specified with no additional options, full Meta-Update tracing is turned on. With `-d` no ARS client tracing is turned on.

With full tracing a great deal of data is generated. Without `-d`, only a very few messages will be traced.

Tracing levels for both Meta-Update and ARS can be specified with the –d: switch options.

```
-d  :  [ fpd2as , ] [ fsap ] [ , file ]
```

The first set of letters specifies the Meta-Update tracing levels. A comma is used to separate the Meta-Update levels and the ARS levels. The second set of letters specifies the ARS client tracing level. A further comma separates these levels from a specific trace file name.

If a full tracing switch is specified, further switches may be specified as the next set of parameters.

For Meta-Update tracing, the levels are specified with a single case sensitive character as follows:

```
S   Severe         Severe error
E   Error          Error
W   Warn           Warning
A   All            Always   like info but never masked out
R   Run            Run   execution instance
  Script Processing    These are on by default but may be turned off.
i   Info           Informational    (on by default)
  Script Debugging     These are echoed when selected with the -d
Q   Qry            ArQuery, Sql;     all query strings
G   Get            ArGet             all ArRecGet ids
U   Put            ArPut             all ArRecPut ids etc
  Debugging settings   These are never echoed.
  Caution:These generate masses of logs and can affect performance.
F   Func           Function entry and exit
d   Dbg            Debugging         detailed debugging
2   Dbg2           Debugging lvl 2   more details yet
a   Data           Data              data values: records, fields
s   Struct         Structure         data Structures
l   List           Script listing and files are logged
```

For ARS tracing, the user id the Meta-Update signs on the update ARS server must be in the Group that the ARS administrator has specified client side logging for in the Server Information panels using the ARS Administrator tool.

The following options can be specified:

s	SQL logging
f	filter logging
a	API logging
p	Plug-in logging

Specifying any ARS tracing implies Meta-Update tracing of level 2.

In the next example, we want the filter traces from ARS and the Meta-Update data traces. This will show us what value each field had before the ARS submit, set, or merge call, as well as the filter logs produced by that call.

```
-d:a,f
```

In this example, we want complete tracing, including complete ARS tracing, and we want to direct it to a specific file:

```
-d:,sfap,d:\trc\my-script.log
```

This has no effect for ServiceNow sessions. Use a double minus d for all ServiceNow transactions and transaction data. No capture of server logs is done.

Two Trace Versions

There are two versions of Meta-Update: one uses local tracing and produces a trace file in the current working directory of where the program is run.

Local Tracing

The local trace file is called `SthMupd.log` unless a file name is specified on the `–d` switch. `SthMupd.log` can be found in the current working directory of the Command Prompt or shell where Meta-Update was run from.

This file is appended to with each execution of Meta-Update. `SthMupd.log` will continuously grow in size. It is recommended that you delete the file before the next execution of Meta-Update.

There is no locking mechanism for multiple instances of Meta-Update running simultaneously in the same directory. This can happen when ARS workflow fires a Meta-Update process on the server.

It is recommended that if Meta-Update will be used in workflow, or in multiple, concurrent instances on a single machine, that the Trace server version be used. The Trace server must be running.

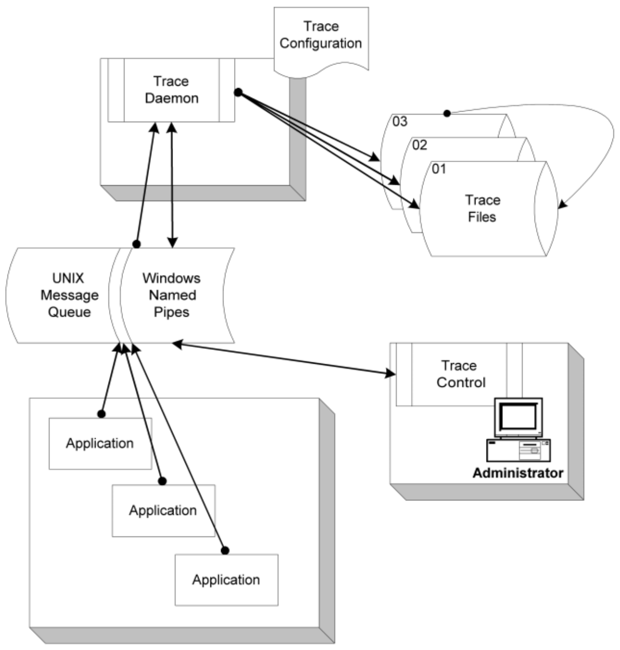
For ad-hoc runs of Meta-Update from a client machine it may be more convenient to use the local trace version.

When using the –d switch, a great deal of logging information may be written.

With or without full tracing, a file is created or appended to each Meta-Update is run. This file will grow in size. It is the user’s responsibility to remove this file from time to time as appropriate.

Server Tracing

An alternative, communication based trace facility is available for high use applications. With this server based trace facility, the machine administrator manages the detail of the messages captured, and the size and number of trace files. Tracing is controlled independently of any application using it.



All client binary (executable) names that have the server based tracing included are suffixed with “Trc”. Meta-Update, for example, would be `SthMupdTrc.exe`.

If the trace daemon is not running, the same local trace file, `SthMupd.log`, is created or appended to. .

The following binaries are supplied with the server based tracing facility.

trcdaem.exe	This is the trace server itself. It should be started automatically when the machine starts.
trcctl.exe	This controls the trace daemon allowing the tracing levels to be set, switching to the next generation of trace file, and shutting down the trace server.
trcecho.exe	This utility adds records to the trace file and can be used in shell scripts or Windows command files.

Note that Meta-Update must be invoked with a `–d` switch for any debug level traces to be sent to the trace daemon. The trace daemon must also be set to capture the level of tracing desired.

The trace daemon uses a configuration file to specify both communication parameters and file handing and other trace daemon operational options.

All trace clients, such as Meta-Update or **`SthMupdTrc.exe`** for example, need to access this file to read the communication parameters. The location of this file is given by an environment variable.

On UNIX the trace daemon uses the POSIX message queue facility. The daemon should be run at a higher priority, or lower nice value, than any of its clients to prevent messages being lost. Further, system parameters should be adjusted so that the message queuing is not a performance bottleneck.

Under normal production usage (without the –d switch) very few messages are sent to the trace daemon and so performance is not generally an issue.

On Windows, Name Pipes are used to implement the inter-process communication. This will generally not require any system parameters to be changed to affect the performance. The trace daemon performance is not generally a bottleneck on Windows systems.

Note that to capture a level of trace messages beyond the minimum, both:

- ✓ The trace daemon is configured to include the desired trace level, or by using the trace control program. the desired trace level is on; and,
- ✓ The program will have been run with the **`-d`** switch specifying the desired trace level.

An environment variable is used by the trace daemon and all trace clients. This environment variable specifies a trace configuration file. The environment variable can be set in Windows as a system wide variable.

```
Set TrcIni=c:\etc\conf\SthTrc.cfg
```

The configuration file must exist. It is an ASCII file (created with Notepad or vi for example) and follows the format rules for a Meta-Update command file but with no section names. It can have these variables:

```
#  Trace facility configuration file for sth-m3
#    file:  e:\etc\conf\trace.ini
#
#    Environment variable must be defined system wide...
#      TrcIni=e:\etc\conf\trace.ini
#

QueueKey   =  e:\etc\conf\trace.ini
TraceFile  =  e:\trc\trace
GenMax     =  99
RecMax     =  500000
TrcLvl     =  dasfp2
TrcTme     =  30
ErrLog     =  e:\trc\error.log

ScOpen     =  cmd /c trcerrm.cmd
```

`QueueKey   =`	is used on Unix platforms only. The message queue is opened using the specified file's i-node as the key. On Windows this parameter is ignored.
`TraceFile  =`	specifies the fully qualified prefix for the trace files. The string specified is suffixed with .xx where xx is the current open trace file.

`GenMax`	=	specifies the maximum number of trace files to produce. Specifying 99, for example, would mean that a maximum of 100 files named e:\trc\trace.01, .02, .. .99 could exist at the same time. After trace.99 is filled up, trace.01 will become the current file.
`RecMax`	=	specifies the maximum number of records per file. When this number is reached, the trace file will be closed and the next trace file will be opened.
`TrcLvl`	=	the starting trace level. See trcctl.exe for more information about the levels and their meanings.
`TrcTme`	=	a normal trace client is presumed to live for a short time between issuing traces. Long lived processes may have larger amounts of time between traces. This specifies the maximum amount of time between calls for the trace daemon to consider that the client program has failed or been aborted without a proper shutdown. When this time is reached, an error trace message will be added to the trace file and client resources will be freed.
`ErrLog`	=	specifies a single file that will collect R and E messages. This file will always grow. It is the administrators responsibility to remove the file on occasion.
`ScOpen`	=	can be used to run a single command file or shell script. It is passed the version number of the file just closed and the fully qualified file name:

In the above example, when the trace switches (say it was on 19 and is now on 20), a command will be run in the system as follows:

```
cmd /c  trcerrm.cmd   19   e:\trc\trace.19
```

See www.softwaretoolhouse.com for more details and for the Trace User document.

Trace Format

A trace record looks like this:

```
hhmmss.nnn f 0pid Prog     text
```

The hhmmss.nnn is the time that the record was created by the application. Note that trace records may appear out of sequence between applications but will never be out of sequence for any one instance of an application. Note also that a single application may have two instances running concurrently.

The f is the highest priority TrcLvl value on the Trc call that sent this trace message. Values are as follows:

```
S   Severe        Severe error
E   Error         Error
W   Warn          Warning
A   All           Always   like info but never masked out
R   Run           Run   execution instance
  Script Processing    These are on by default but may be turned off.
I   Info          Informational   (on by default)
  Script Debugging     These are echoed when selected with the -d
Q   Qry           ArQuery, Sql;     all query strings
G   Get           ArGet             all ArRecGet ids
U   Put           ArPut             all ArRecPut ids etc
  Debugging settings   These are never echoed.
  Caution:These generate masses of logs and can affect performance.
F   Func          Function entry and exit
d   Dbg           Debugging         detailed debugging
2   Dbg2          Debugging lvl 2   more details yet
a   Data          Data              data values: records, fields
s   Struct        Structure         data Structures
```

The 0pid is the process identifier, in hexadecimal, of the process that generated the trace message. This number can be used to select the trace records for a specific instance of a specific application.

The Prog is the program name coded on the application's TrcInit call. Each application that uses the trace facility should document its use of the facility in its User's Guide. You can use this field to extract those records written by any one application.

The text is the actual text of the trace message and is entirely application dependent.

Firing from Workflow

Meta-Update may be fired from workflow as Run Process or Set Fields $PROCESS$ filter or active link.

When firing from workflow on the server, the environment is that of the ARS server process. It is prudent to code a script or batch file in the workflow and then have that script or batch file set up the environment for the run, invoke Meta-Update, and possibly do some termination activities.

The environment generally includes a path to the executable and to any required shared libraries or dlls, other environment variables, parameters, and the working directory.

As workflow is fired at independent times, it is possible for multiple copies of Meta-Update to be running simultaneously. If so, the Server based tracing version is highly recommended to properly serialise log files.

Developing Scripts

Normal Meta-Update runs will report script errors with an ‘E’ level message echoed to the console. That message will print the script file name, section, line number, and, if appropriate, the keyword being processed.

```
114159.531 E [Do] [asg-init] AssignInit apply was aborted in file:
                 FD-SupGrp-Ren.ini  [asg-init]   @Cmd=  line: 74
```

Errors may be caused by different things:

- Syntax errors
- ARS reported errors such as unrecognised schema names or field names or labels
- LookUp or Load failures
- User Aborts

Meta-Update has several switches that will aid in script development which would normally not be used in production runs.

-e	single Error	With this switch, any error in any section will stop the run. We recommend you use this switch when you develop and test scripts. You will generally not want it on production runs.
-v	Verbose	This prints all query qualifications and results to the console and to the log file. We recommend you use this switch when you develop and test scripts. You will generally not want it on production runs.
-n	null	This switch prevents any ARS updates or creates. This is only useful for the most simple of scripts as generally launched sections depend on access to a previous sections updated and reread record reference.
-d	Logging: Debug	This should not normally be needed. It is intended to be used when using Meta-Update support. It provides complete debug level information on the job and generates masses of logs. You can also specify you want ARS client logging with this switch. See ***Tracing*** above for more information.
-g	Script Debugger	This invokes the Meta-Update script debugger. The script debugger allows you to set breakpoints and single step though your script’s operation. You can get debugging help, print your script, examine references, control breakpoints, and resume normal execution. See **Script Debugging** below for more information about using the Meta-Update Script Debugger.

In this example, a script **`Abort=`** was set by an **`AssignInit=`** section that ensured there was at least one matching Support Organisation.

Another example where a bad value is passed as a script argument:

```
E:\> SthMupd -v -e FD-SupGrp-Ren.ini Do -Org "Help Desk"
```

The **`-v`** switch echoes the exact query qualifications sent to the Remedy Server

The script issues several “E” messages and then an abort.

Meta-Update tells you the script issued an Abort.

Meta-Update

```
Meta-Update    Version 5.56 (x64) for ARS lib 8.1.2
           (c) Copyright 1996-2015 by Software Tool House Inc.
               www.softwaretoolhouse.com
114159.515 q [Do] QuerySql: Svr: sthv1
114159.515 q [Do] QuerySql: Qualification: : 0000: select count(*) from
             CTM_Support_Group where  Support_Organiza
114159.515 q [Do] QuerySql: Qualification: : 0040: tion = 'Qelp Desk'
114159.515 q [Do] QuerySql: returned 1 records of 1.
114159.515 i [Do] Msg: Found 0 records with:  'Support Organization' == "Qelp Desk"
114159.515 E [Do] Msg: The Support Organisation argument must match 1 or more records
             of CTM:Support Group"
114159.515 E [Do] Msg: Please check the spelling of your command line argument."
114159.531 E [Do] Abort: ..aborting."
114159.531 E [Do] [asg-init] AssignInit apply was aborted in file:  FD-SupGrp-Ren.ini
             [asg-init]   @Cmd=  line: 74
114159.531 E IniRdo of FD-SupGrp-Ren.ini [Do] failed with 3 - ArPutIini: Parm error 3
114159.531 i Statistics:
114159.531 i         Sections:                   1
114159.531 i         Maximum section depth:      1
114159.531 i         Output Schemas:             0
114159.531 i         Output Schema records:      0   created
114159.531 i         Output Schema records:      0   updated  (with 0 skipped)
114159.546 i         Outputs OK:                 0
114159.546 i         Outputs Errors:             0
114159.546 i         Outputs Aborts:             0
114159.546 i         Input   Errors:             0
114159.546 E error: some errors occurred.  Check for errors above this message.
114159.546 E terminating unsuccessfully in 0 sec.
```

In this next example, the script file's **`Query=`** at line 65 referenced a ReadServer tag which was not defined as the script didn't need use additional servers.

```
Query       = @Itsm6, User, User
```

Source line in error.

```
E:\> SthMupd  QQQ-TblRpt-User.ini Do sthv1 Demo -start 1 -max
10
```

Error: Server reference not found.

Script line number in error.

```
Meta-Update    Version 5.56 (x64) for ARS lib 8.1.2
           (c) Copyright 1996-2015 by Software Tool House Inc.
               www.softwaretoolhouse.com
113416.785 i [Do] One:
113416.785 i [Do] One: Launching:  1 of  1 [Do1]
113416.785 E [Do] One: FlIniFindCtl: Server Tag: Itsm6 not found
113416.785 E [Do] One: ArIiniQuery: FlIniRefFindCtl for Itsm6 failed at file:  QQQ-
             TblRpt-User.ini  [Do1]   Query=  line: 65
113416.785 E [Do] One: ArPutIiniRinit: ArIiniQuery failed (rc=4) in file:  QQQ-TblRpt-
             User.ini  [Do1]   Query=  line: 65
113416.785 E [Do] One: ArPutIiniRinit for Do1 returned 3 - ArPutIini: Parm error 3
113416.785 E [Do] One: ArPutIiniRdo: DoLaunch failed!
113416.801 E [Do] One: 0 record OK; 1 records with errors; total: 1.
113416.801 E IniRdo of QQQ-TblRpt-User.ini [Do] failed with 3 -
113416.801 i Statistics:
113416.801 i         Sections:                   1
113416.801 i         Maximum section depth:      1
113416.801 i         Singleton Sections:         1   errors:      0
113416.801 i         Output Schemas:             0
113416.801 i         Output Schema records:      0   created
113416.801 i         Output Schema records:      0   updated  (with 0 skipped)
113416.801 i         Outputs OK:                 0
113416.817 i         Outputs Errors:             0
113416.817 i         Outputs Aborts:             0
113416.817 i         Input   Errors:             0
113416.817 E error: some errors occurred.  Check for errors above this message.
113416.817 E terminating unsuccessfully in 0 sec.
```

Sample Scripts

Samples

The following sample scripts can be used as learning vehicles and are included in the distribution package. The distribution may be downloaded from the web.

If you are new to Meta-Update scripting, start with less complex scripts. Some scripts are copies of simpler scripts with an addition that adds functionality and complextity.

A good idea is to open the script in an editor and single step through the script using the debugger.

Samples List

Script	What it does	Com-plexity 0 .. 10	What it shows
100-Path	List all path elements	0	Loops through the Path directories either listing them or creating a CSV.
110-PathFind	Find a file along a path - like Linux's "which"	1	Based on the above, shows use of Until= and spawing a client process.
000-SvrInfo	Make a CSV of all Server Info values	0	Simplist of scripts, Loops through all fields of the predefined tag ARS_INFO making a CSV.
000-SvrInfo-RdSvr	Make a CSV of all Server Info values coming from a second session	0	Identical to the above but also shows opening two server sessions.
005-ArSchema	Make a CSV of a query (or all) arschema tables with record and workflow count columns	2	Demonstrates QuerySql= used in an Iteration and in LookUp to count records, Active Links, Filters, Guides. Demonstrates Output= to create a CSV report. Will throw an error on a pre 7.1 ARS Server.
006-ArSchema-pre71	As above but for servers without the "Viewname" column.	3	As above, but includes a complex bit of assignment logic to get an SQL ViewName for servers before 7.1 when the column was added to arschema. Will also work against a post-7.1 server.
600-ItsmVer	Display ITSM version	0	A script with no iterations doing a single SQL Query as a LookUp.
610-App-Prop	Make a CSV of SHARE:Application_Properties filling in the Display Only Application Name column.	1	Simple Query on a single table with a copy to a file and an explicit assignment using an SQL Query

900-SwLogs	Switch server logs files and set **`DEBUG_MODE`**	1	Demonstrates an **`Update=`** and an assignment to **`AR_INFO`**, **`DEBUG_MODE`**. Functions by writing to a vendor form introduced in 7.1
910-SvrInfo-set	Set a single Server Info value (like Admin Mode)	0	Very powerful, yet the simplist of scripts, only a single Assignment statement setting the value specified. Caution: sets dynamic server settings like admin mode, mid-tier passwords, etc.
920-Svr-HostName-Change	Set all values needed on a host name change or VM replication. Use after all config file changes are made and the server is running.	2	Demonstrates Query=, Update=, Launching a sequence of disparate sections to update a set of tables.
320-Tbl-Bkp	Backup an ARS table to CSV (with renamed attachments)	4	Query=, Output=, Loop= Fields. Saving attachments to the file system.
620-Tbl-Rst	Restore from a CSV to an ARS table t(with attachments)	4	Query=, Output=, Loop= Fields. Saving attachments to the file system.
340-Tbl-All-Bkp	Backup a set of ARS tables to a set CSV files (with renamed attachments)	6	Query=, Output=, Loop= Fields. Saving attachments to the file system.
460-Change-Approve	Approve a set of Changes and optionally move them to the next stage.	6	Shows how a single script can run off three different inputs: a file, a list, or a query, then progress to the same section to effect one or two table updates.

Descriptions

100-Path.ini

This simple script lists or creates a CSV of one column listing the paths in any path-like environment variable..

What it does	List all path elements.
What it shows	Loops through all fields of the predefined tag ARS_INFO optionally making a CSV.
Description	This is a good beginners' script. It does a string loop and shows how to assign a double referenced value – the environment variable when passed on the command. The next script, 110-PathFind is an enhancement to this script that finds a specific file along the path.
File location	`samples\000-Misc\`

Command Line	`SthMupd 100-Path.ini Do -go`
	`[ -var EnvVarName ]`
	`[ -fout output.csv ]`

110-PathFind.ini

This script is based on 100-Path.ini. It loops through the path strings and spawns a "dir" or "ls" command to look for a file along that path. If it finds the file, it stops the loop.

What it does	Find a file along a path.
What it shows	Loops through all fields of the predefined tag ARS_INFO optionally making a CSV.
Description	Shows use of Until= to limit an iteration. Shows spawing a client processes.
File location	`samples\000-Misc\`

Command Line	`SthMupd 110-PathFind.ini Do`
	`-ptn file_name`
	`[ -var EnvVarName ]`

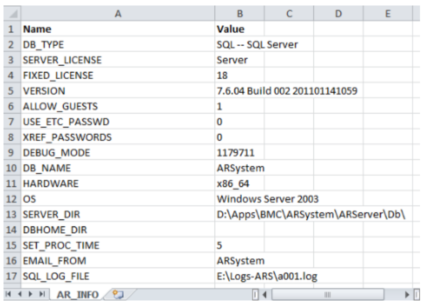
Examples

```
SthMupd  110-PathFind.ini  Do  -ptn  SthMupd.exe
SthMupd  110-PathFind.ini  Do  -ptn  500-Arch.ini
                               -var  SthScriptPath
```

000-SvrInfo

This script loops through the path strings and spawns a “dir” or “ls” command to look for a file along that path. If it finds the file, it stops the loop. It is useful to attach to a BMC ticket. The script simply loops through the predefined AR_INFO Tag and outputs a CSV file.

What it does Creates a CSV of all AR_INFO fields (Server Information)..



	A	B	C	D	E
1	**Name**	**Value**			
2	DB_TYPE	SQL -- SQL Server			
3	SERVER_LICENSE	Server			
4	FIXED_LICENSE	18			
5	VERSION	7.6.04 Build 002 201101141059			
6	ALLOW_GUESTS	1			
7	USE_ETC_PASSWD	0			
8	XREF_PASSWORDS	0			
9	DEBUG_MODE	1179711			
10	DB_NAME	ARSystem			
11	HARDWARE	x86_64			
12	OS	Windows Server 2003			
13	SERVER_DIR	D:\Apps\BMC\ARSystem\ARServer\Db\			
14	DBHOME_DIR				
15	SET_PROC_TIME	5			
16	EMAIL_FROM	ARSystem			
17	SQL_LOG_FILE	E:\Logs-ARS\a001.log			

AR_INFO

What it shows Shows a “Fields Loop” on the predefined tag AR_INFO. Shows a two-column CSV output= creation.

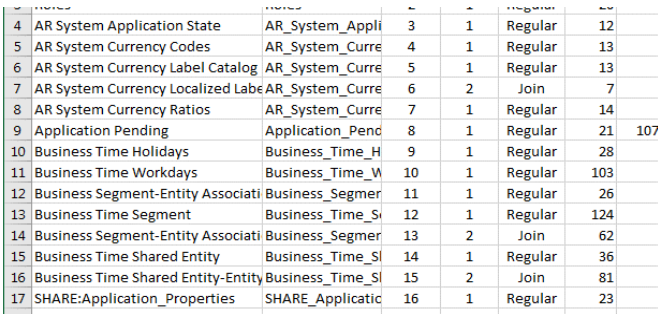
Description This is a very simple beginners’ script. It does a fields loop and Output= to create the CSV..

File location **`samples\ 003-SvrInfo\`**

Command Line **`SthMupd 000-SvrInfo.iniDo -outf MyServerInfo.csv`**

005-ArSchema – AR Schema Report

This simple script creates a CSV of the tables in an ARS server with additional columns for and the number of records they contain.



4	AR System Application State	AR_System_Appli	3	1	Regular	12	
5	AR System Currency Codes	AR_System_Curre	4	1	Regular	13	
6	AR System Currency Label Catalog	AR_System_Curre	5	1	Regular	13	
7	AR System Currency Localized Labe	AR_System_Curre	6	2	Join	7	
8	AR System Currency Ratios	AR_System_Curre	7	1	Regular	14	
9	Application Pending	Application_Pend	8	1	Regular	21	107
10	Business Time Holidays	Business_Time_H	9	1	Regular	28	
11	Business Time Workdays	Business_Time_W	10	1	Regular	103	
12	Business Segment-Entity Associati	Business_Segmer	11	1	Regular	26	
13	Business Time Segment	Business_Time_S	12	1	Regular	124	
14	Business Segment-Entity Associati	Business_Segmer	13	2	Join	62	
15	Business Time Shared Entity	Business_Time_Sl	14	1	Regular	36	
16	Business Time Shared Entity-Entity	Business_Time_Sl	15	2	Join	81	
17	SHARE:Application_Properties	SHARE_Applicatic	16	1	Regular	23	

What it does It does an SQL Query the **`arschema`** table, does a few select count(*) as LookUps, and generates a CSV.

What it shows Shows **`QuerySql=`** used in an Iteration and in LookUps to count records, Active Links, Filters, Guides.

Shows **`Output=`** to create a CSV file.

Description This is a very simple beginners’ script. It is a single section that iterates through a **`QuerySql=`** and **`Output=`**.The **`Output=`** assignments use **`QuerySql=`** in **`LookUp=`** for the counts.

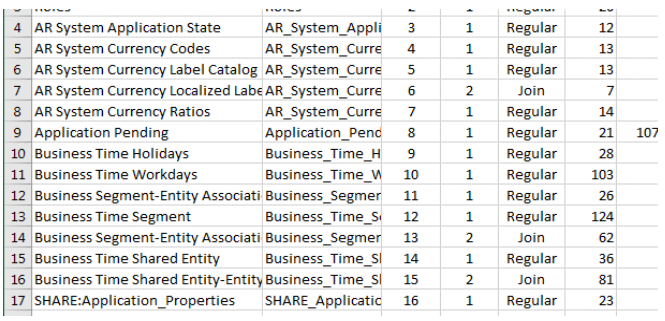
File location **`samples\003-SvrInfo\`**

Command Line

```
SthMupd  005-ArSchema.ini  Do -outf  arschema.csv
SthMupd  005-ArSchema.ini  Do -outf  arschema-CS.csv
                              -ptn   ”BMC.CORE:%”
```

006-ArSchema-pre71 – AR Schema Report

This is identical to the above but one of two sections are launched based on the ARS Server version. When run against a pre ARS 7.1 server, the script itself assigned the "View Name" field as the arschema table does not have that column.



4	AR System Application State	AR_System_Appli	3	1	Regular	12	
5	AR System Currency Codes	AR_System_Curre	4	1	Regular	13	
6	AR System Currency Label Catalog	AR_System_Curre	5	1	Regular	13	
7	AR System Currency Localized Labe	AR_System_Curre	6	2	Join	7	
8	AR System Currency Ratios	AR_System_Curre	7	1	Regular	14	
9	Application Pending	Application_Pend	8	1	Regular	21	107
10	Business Time Holidays	Business_Time_H	9	1	Regular	28	
11	Business Time Workdays	Business_Time_W	10	1	Regular	103	
12	Business Segment-Entity Associati	Business_Segmer	11	1	Regular	26	
13	Business Time Segment	Business_Time_S	12	1	Regular	124	
14	Business Segment-Entity Associati	Business_Segmer	13	2	Join	62	
15	Business Time Shared Entity	Business_Time_Sl	14	1	Regular	36	
16	Business Time Shared Entity-Entity	Business_Time_Sl	15	2	Join	81	
17	SHARE:Application_Properties	SHARE_Applicatic	16	1	Regular	23	

What it does As 005-ArSchema.

What it shows Shows a complex bit of assignment logic to "calculate" an SQL ViewName depending on the Remedy table name, its Schema Id, the server's database type.

Description This script is identical to the above but the main section launches one of two sections for pre and post ARS 7.1 and the ViewName value, either from the arschema table (post 7.1) or derived in the script (pre 7.1).

This script is not documented in the user guide and is left for the reader to explore..

File location `samples\003-SvrInfo\`

Command Line

```
SthMupd  006-ArSchema.ini  Do -outf  arschema.csv
SthMupd  006-ArSchema.ini  Do -outf  arschema-CS.csv
                              -ptn    "BMC.CORE:%"
```

600-ItsmVer

This simplest of scripts (5 lines) displays the ITSM Version by using a QuerySql= in a LookUp.

What it does It does an SQL Query on SHARE:Application Properties for a specific key / name and issues a message.

What it shows Shows a `QuerySql=` used in a LookUp and the simplest of Iteration Sections, a single AssignInit.

Description This is a very simple beginners' script. It is a single section that has only an `AssignInit=` and that assignment sectionhas two statements, one to LookUp the version, and one to display it.`Output=`.The `Output=` assignments use `QuerySql=` in `LookUp=` for the counts.

File location `samples\003-SvrInfo\`

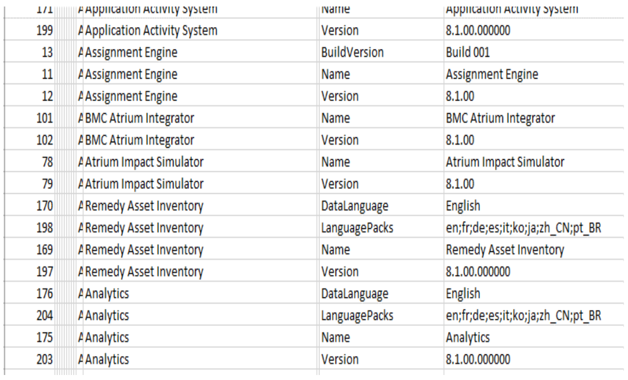
Command Line

```
SthMupd  600-ItsmVer.ini  Do -go
```

610-ItsmAppProp

Make a CSV of SHARE:Application_Properties filling in the Display Only Application Name column..

What it does It does a Query on SHARE:Application Properties and does a cached **`LookUp`** for the Application Name.



171	A	Application Activity System	Name	Application Activity System
199	A	Application Activity System	Version	8.1.00.000000
13	A	Assignment Engine	BuildVersion	Build 001
11	A	Assignment Engine	Name	Assignment Engine
12	A	Assignment Engine	Version	8.1.00
101	A	BMC Atrium Integrator	Name	BMC Atrium Integrator
102	A	BMC Atrium Integrator	Version	8.1.00
78	A	Atrium Impact Simulator	Name	Atrium Impact Simulator
79	A	Atrium Impact Simulator	Version	8.1.00
170	A	Remedy Asset Inventory	DataLanguage	English
198	A	Remedy Asset Inventory	LanguagePacks	en;fr;de;es;it;ko;ja;zh_CN;pt_BR
169	A	Remedy Asset Inventory	Name	Remedy Asset Inventory
197	A	Remedy Asset Inventory	Version	8.1.00.000000
176	A	Analytics	DataLanguage	English
204	A	Analytics	LanguagePacks	en;fr;de;es;it;ko;ja;zh_CN;pt_BR
175	A	Analytics	Name	Analytics
203	A	Analytics	Version	8.1.00.000000

What it shows Shows a **`Query=`** used with an **`Output=`** in an iteration section, and a **`QuerySql=`** used in a **`LookUp`** in the Output assignments.

Description This script is a single section using a **`Query=`** and an **`Output=`** is a common pattern. The assignments are copied from the queried record into the output record and added fields are filled in with a **`LookUp`**.

File location **`samples\003-SvrInfo\`**

Command Line **`SthMupd`** 610-ItsmAppProp.ini **`Do -outf DevSvrAppPropt.csv`**

900-SwLogs

Turns off server logging, switches server logs files, and then sets **`DEBUG_MODE`** to turn on logging again.

What it does It write to the vendor form introduced in ARS 7.1 that controls the server settings to set all log files, and then sets **`DEBUG_MODE`** on SHARE:Application Properties for a spefic key / name and issues a message.

What it shows A simple Update= with no Query= and setting the AR_INFO, DEBUG_MODE to control the server.

Description This is a very simple beginners' script. It is a single section that has only an **`AssignInit=`** and that assignment sectionhas two statements, one to LookUp the version, and one to display it.**`Output=`**.The **`Output=`** assignments use **`QuerySql=`** in **`LookUp=`** for the counts.

File location **`samples\003-SvrInfo\`**

Command Line **`SthMupd`** 900-SwLogs.ini **`Do -off`**
`SthMupd` 900-SwLogs.ini **`Do -log Bug41`**

910-SvrInfo-set

Set a single Server Info value (like Admin Mode).

What it does	Very powerful, yet the simplist of scripts: only a single Assignment statement setting the value specified.
Caution:	Sets dynamic server settings like admin mode, mid-tier passwords, etc.
What it shows	A simple **`AssignInit=`** with a single assigment setting the **`AR_INFO`** value specified.
Description	This is a very simple beginners' script. It is a single section that has only an **`AssignInit=`** and that assignment section has one assignment.
File location	**`samples\003-SvrInfo\`**

Command Line

```
SthMupd 910-SvrInfo-set.ini Do -key DEBUG_MODE
                               -val   0
```

320-Tbl-Bkp

Backup an ARS table to a CSV file extracting all attachments to the file system using file names based on Request IDs.

What it does	A small, powerful script that saves the contents of an ARS table as a CSV file. It also extracts any attachments by saving them with the Request ID in the file name.
What it shows	A simple **`Query=`** with a **`Output=`** creating as many CSV rows as records returned from the Query. Also shows a Launch that does a **`Loop= Fields`** through any non-null attachment fields.
Description	This is only the next step above a beginners' script. It has a single section that performs the backup and Launches a second section to extract any attachments.
File location	**`samples\003-SvrInfo\`**

Command Line

```
  SthMupd 310-Tbl-Bkp.ini Do
   -schema      ARS-Table-Name
   -Fout        Output-CSV-file
 [ -qry         "Query-Text" ]
```

620-Tbl-Rst

Backup an ARS table to a CSV file extracting all attachments to the file system using file names based on Request IDs.

What it does	A companion script to 320-Tbl-Bkp. Restoores contents of a CSV to a table including any saved attachments.
What it shows	A simple **`File=`** with an Update= creating/updating as many ARS records as CSV rows. Also shows a Launch that does a Loop= Fields through any non-null attachment fields.
Description	This is only the next step above a beginners' script. It has a single section that performs the backup and Launches a second section to extract any attachments.
File location	`samples\003-SvrInfo\`

Command Line

```
  SthMupd 610-Tbl-Rst.ini Do
   -schema      ARS-Table-Name
   -inpf        Output-CSV-file
[  -qry         "Query-Text"  ]
```

340-Tbl-All-Bkp

Backup a set of ARS tables to a set of CSV files extracting all attachments to the file system using file names based on Request IDs.

This is an enhancement to **`320-Tbl-Bk`**p.

What it does	A companion script to 320-Tbl-Bkp. Restoores contents of a CSV to a table including any saved attachments.
What it shows	A simple **`File=`** with an Update= creating/updating as many ARS records as CSV rows. Also shows a Launch that does a Loop= Fields through any non-null attachment fields.
Description	This is only the next step above a beginners' script. It has a single section that performs the backup and Launches a second section to extract any attachments.
File location	`samples\003-SvrInfo\`

Command Line

```
  SthMupd 610-Tbl-Bkp.ini Do
   -schema      ARS-Table-Name
   -Fout        Output-CSV-file
[  -qry         "Query-Text"  ]
```

460-Change-Approve

Input is a CSV of Changes that are approved. This script processes that input, ensuring Changes are in Scheduled for Approval status, approving the changes, and optionally, moving them to their next phase.

This was a Meta-Update Proof-of-Concept script that took a total of 4 hours to create. This single script was a 100% ROI for Meta-Update.

What it does

Processes an input CSV of Change Request numbers and approves these Changes.

What it shows

Shows how to make the same script operate on different inputs: in this case, a File of Change Requests, a List, or a Query.

A **`File=`**, **`Loop=`**, or **`Query=`** are used to select the Changes that are in Status: Scheduled for Approval.

The script throws an error if a selected Change is not in the correct Status.

The script now calls a single section that adds or updates a signatire record.

Then, it updates a Signature-Change Join record to validate the process.

Description

This script needs some configuration changes. It is provided as a practical examples of batch processing possible with Meta-Update.

File location

`samples\430-ITSM-Chg\`

Command Line

```
  SthMupd 460-Change-Approve.ini Do
[   -list     CRQ000000000119 [, … ] ]
[   -Finp     input-file             ]
[   -qry      "Query-Text"           ]
```

Closed Ticket Duplicator

Real Customer Problem

Development time:
***three* hours!**

A mail robot must not reopen a ticket, nor attach an email to a closed ticket.

This ticket replicator creates a new ticket, with the salient data from the old ticket, assigning it to the last group that closed the old ticket, replicating all emails and other associated records, and finally linking the two tickets together for the GUI button.

This script demonstrates launching other sections so that multiple tables are processed.

Server data extract

Real Customer Problem

A single customer has many locations, people, services, etc. This script is used to copy a single customer’s data from production to development for a single developer replacing any customer contact information with the developer’s information.

Development time:
***three* hours!**

This was used in a large development team of a bespoke telecoms client to facilitate development and testing.

Server delta copy

A simple script copying all changed records from one server to another – say a read only, reporting server..

Development time:
***one* hour!**

Demonstrates using Read Servers, QuerySql, Merge, Query, Update, the Copy assignment command.

Ticket Creation Batch Command

A simple script that creates a ticket accepting different command line parameters.

Development time:
***one* hour!**

This script demonstrates the simple creation of a record based on command line arguments. It introduces the common elements of a Meta-Update script.

100-Path

This script simply writes the components of the PATH environment variable to a single column "CSV" file or to the console as messsages.

It performs no ARS queries or updates at all.

The script demonstrates:

- How to use a Loop = String statement.
- How to reference a value when the reference field is itself a reference.
- How to use Output= to create a CSV

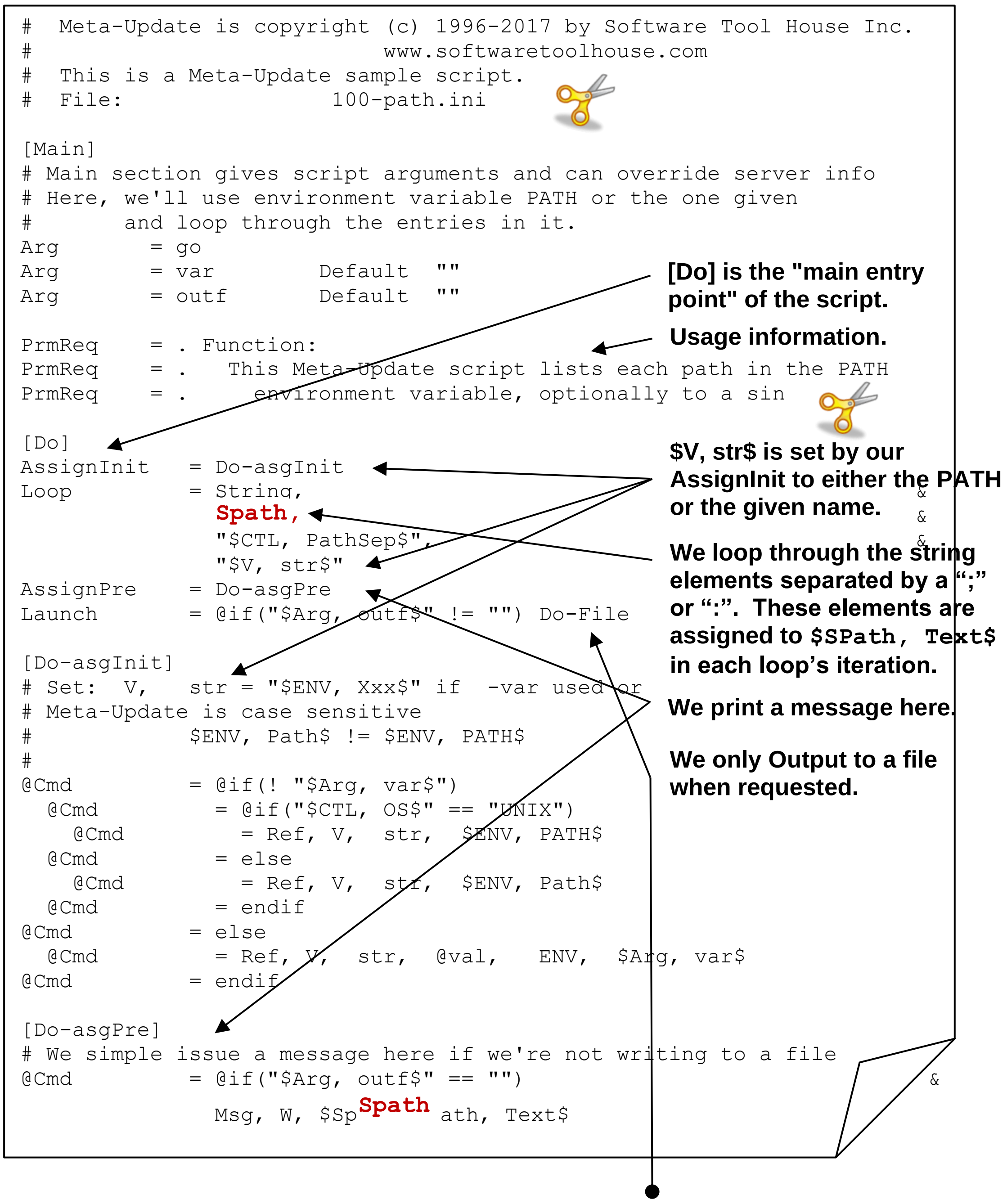
Usage Instructions

```
. Function:
.   This Meta-Update sample script simply lists each path in
the
      PATH or other, environment variable,
      optionally to a single column CSV file.
.
. Usage
.   SthMupd   100-Path  Do  -go
.                           -outf  out-file
.                           -var   Path
.
.     where       -go          is ignored but needed to avoid
                                 help display with no arguments
.                 -var         can be any path-like ENV var,
.                                "SthScriptPath" for example
.                 -outf        will cause output to a file
.                               (else console)
.
. Examples
.   SthMupd   100-Path  Do  -outf c:MyDatapathinfo.txt
.   SthMupd   100-Path  Do
.
```

Sample Output

```
>> SthMupd.exe 100-Path.ini Do -go:
Meta-Update    Version 5.74 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2017 by Software Tool House Inc.
               www.softwaretoolhouse.com
W [Do] Lp:   1 of 43: Msg: d:\Apps\Sth\Meta-Update\msch\
W [Do] Lp:   2 of 43: Msg: d:\Apps\Sth\Meta-Update\mdel\

W [Do] Lp:  43 of 43: Msg: C:\Program Files\Common
Files\Intel\WirelessCommon\
i Statistics:
i        Sections:                    1
i        Maximum section depth:       1
i        Loops:                       1
i        Loop values:                43   errors:      0
i terminating successfully in 2 sec.
```

Development time:
under fifteen minutes!



```
#   Meta-Update is copyright (c) 1996-2017 by Software Tool House Inc.
#                        www.softwaretoolhouse.com
#   This is a Meta-Update sample script.
#   File:                100-path.ini

[Main]
# Main section gives script arguments and can override server info
# Here, we'll use environment variable PATH or the one given
#       and loop through the entries in it.
Arg         = go
Arg         = var         Default  ""
Arg         = outf        Default  ""

PrmReq      = . Function:
PrmReq      = .   This Meta-Update script lists each path in the PATH
PrmReq      = .     environment variable, optionally to a sin

[Do]
AssignInit     = Do-asgInit
Loop           = String,
                 Spath,
                 "$CTL, PathSep$",
                 "$V, str$"
AssignPre      = Do-asgPre
Launch         = @if("$Arg, outf$" != "") Do-File

[Do-asgInit]
# Set:  V,    str = "$ENV, Xxx$" if  -var used or
# Meta-Update is case sensitive
#              $ENV, Path$ != $ENV, PATH$
#
@Cmd           = @if(! "$Arg, var$")
  @Cmd           = @if("$CTL, OS$" == "UNIX")
    @Cmd           = Ref, V,   str,   $ENV, PATH$
  @Cmd           = else
    @Cmd           = Ref, V,   str,   $ENV, Path$
  @Cmd           = endif
@Cmd           = else
  @Cmd           = Ref, V,   str,   @val,   ENV,   $Arg, var$
@Cmd           = endif

[Do-asgPre]
# We simple issue a message here if we're not writing to a file
@Cmd           = @if("$Arg, outf$" == "")
                 Msg, W, $SpSpath ath, Text$
```

[Do] is the "main entry point" of the script.

Usage information.

$V, str$ is set by our AssignInit to either the PATH or the given name.

We loop through the string elements separated by a ";" or ":". These elements are assigned to `$SPath, Text$` in each loop's iteration.

We print a message here.

We only Output to a file when requested.

```
[Do-File]
# We're writing to a file
Output          = F,
                  File-Def,
                  $Arg, outf$
Assign          = Do-File-asg

[Do-File-asg]
#      For the single output "record" we just have one field
Path            = Spath,  Text


[File-Def]
#      This defines the file as a single column CSV.
#
Type            = Delimited, ",", FldHdr
Format          = Csv
Fields          = File-Def-Flds

[File-Def-Flds]
Path            = $
```

We only Output to a file when requested.

We write the single value which our `Loop`= seter the PATH or the given name.

The file is defined as a single column CSV.

110-PathFind

This script is based on 100-Path.ini. It is enhanced to find a file along a Path.

An argument is added: the file to find. A client process is added: the “dir” or “ls” in the path element. An Until= is added: to halt processing when the file is found.

It performs no ARS queries or updates at all.

The script demonstrates:

- How to use a **`Loop=`** String statement.
- How to reference a value when the reference field is itself a reference.
- How to use **`Until=`** to limit a section’s iteration
- How to use a **`Spawn`** reference command and process the results

Usage Instructions

```
      Function:
.    This Meta-Update sample script finds a file along a PATH or
.      Path-like environment variable
.
. Usage
.    SthMupd   110-PathFind  Do  -ptn  file   [ -var  "PATH"   ]
.
.
.      where        -ptn        is a required file name
.                   -var        is an optional Env variable to use
.                                  default: Path
.
. Examples
.    SthMupd   110-PathFind  Do  -ptn SthMupd.exe
.    SthMupd   110-PathFind  Do  -ptn 500-Arch.ini -var SthScriptPath
.
```

Sample Output

```
>> SthMupd.exe 110-PathFind.ini Do -ptm SthMupd.exe
Meta-Update    Version 5.74 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2017 by Software Tool House Inc.
               www.softwaretoolhouse.com
W [Do] Lp: 1 of 43: Spawn process returned 1 for: dir
d:\Apps\Sth\Meta-Update
\msch\msch_x64_a910_d_trc\mupd.exe
W [Do] Lp: 1 of 43: Spawn process returned 1 for:
             dir  d:\Apps\Sth\Meta-Update\msch\SthMupd.exe
W [Do] Lp:  2 of 43: Spawn process returned 1 for:
             dir  d:\Apps\Sth\Meta-Update\mdel\SthMupd.exe
i [Do] Lp:  3 of 43: Until= condition taken on iteration: 3 at
             110-PathFind.ini [Do] Until line: 61.
i [Do] Lp:  3 of 43: Msg: .
i [Do] Lp:  3 of 43: Msg: .
i [Do] Lp:  3 of 43: Msg: mupd.exe found in:
                          d:\Apps\Sth\Meta-Update \SthMupd\
i [Do] Lp:  3 of 43: Msg: .
i [Do] Lp:  3 of 43: Msg: .
i Statistics:
i         Sections:                    1
i         Maximum section depth:       1
```

```
i          Loops:                          1
i          Loop values:                    3    errors:      0

i terminating successfully in 2 sec.
```

Development time:
under fifteen minutes!

```
#   Meta-Update is copyright (c) 1996-2017 by Software Tool House Inc.
#                         www.softwaretoolhouse.com
#   This is a Meta-Update sample script.
#   File:                110-PathFind.ini

[Main]
# Main section gives script arguments and can override server info
Arg          = ptn
Arg          = var        Default  ""

PrmReq      = . Function:
PrmReq      = .   This Meta-Update script finds a file along a PATH or
PrmReq      = .     path-like environment variable.

[Do]
AssignInit     = Do-asgInit
Loop           = String,                                                  &
                 Spath,                                                   &
                 "$CTL, PathSep$",                                        &
                 "$V, str$"
Until          = @if(! "$V, rc$")
AssignPre      = Do-asgPre
AssignTerm     = Do-asgTerm

[Do-asgInit]
# Set:  V,     str = "$ENV, Xxx$" if  -var used or
#              $ENV, Path$ ($ENV, PATH$)
#
@Cmd           = @if(! "$Arg, var$")
  @Cmd           = @if("$CTL, OS$" == "UNIX")
    @Cmd           = Ref, V,   str,   $ENV, PATH$
  @Cmd           = else
    @Cmd           = Ref, V,   str,   $ENV, Path$
  @Cmd           = endif
@Cmd           = else
  @Cmd           = Ref, V,   str,   @val,   ENV,   $Arg, var$
@Cmd           = endif

[Do-asgPre]
# We spawn a "dir" or "ls" process to find the file
@Cmd           = @if("$CTL, OS$" == "UNIX")
  @Cmd           = Ref, V,   Cmd,   "ls -l $Spath, Text$/$Arg, ptn$"
@Cmd           = else
  @Cmd           = Ref, V,   Cmd,   "dir  $Spath, Text$\\$Arg, ptn$"
@Cmd           = endif
@Cmd           = Ref, V,   @spawn,   $V, Cmd$
@Cmd           = Ref, V,   dir,      $Spath, Text$

[Do-asgTerm]
@Cmd           = Msg, I,   .
@Cmd           = @if("$V, rc$")
  @Cmd           = Msg, I,   $Arg, ptn$ not found along ENV $Arg, var$
@Cmd           = else
  @Cmd           = Msg, I,   $Arg, ptn$ found in: $V, dir$
@Cmd           = endif
```

[Do] is the "main entry point" of the script.

Usage information.

$V, str$ is set by our AssignInit to either the PATH or the given name.

We loop through the string elements separated by a ";" or ":". These elements are assigned to `$SPath, Text$` in each loop's iteration.

The `Until=` breaks the loop when a file is found.

In each iteration, the AssignPre spawns a "dir" or "ls" command.

This prints the results (found or not) after the section completes.

`Spawn`, sets `rc`, `stdout`. `stderr` in the Tag "V".

The Tag `Spath`, is not available when the section ends and must be saved.

000-SvrInfo

This simple script outputs a CSV containing the fields and values of the predefined **AR_INFO** Tag.

The **AR_INFO** Tag is automatically defined for every Meta-Update script and is the ARS Server Information. You can use it to determine the database type, the server version, or any of hunderds of dynamic server information.

This script is very useful for ansering a BMC Ticket's query of "Server Environment". Run the script and attach the output file to the Incident, for complete and accurate information about your server environment.

A single argument is needed to specify the output file. This script performs no ARS queries or updates at all.

The script demonstrates:

- How to use a **Loop=** Fields statement.
- How to use an Output= to create a CSV

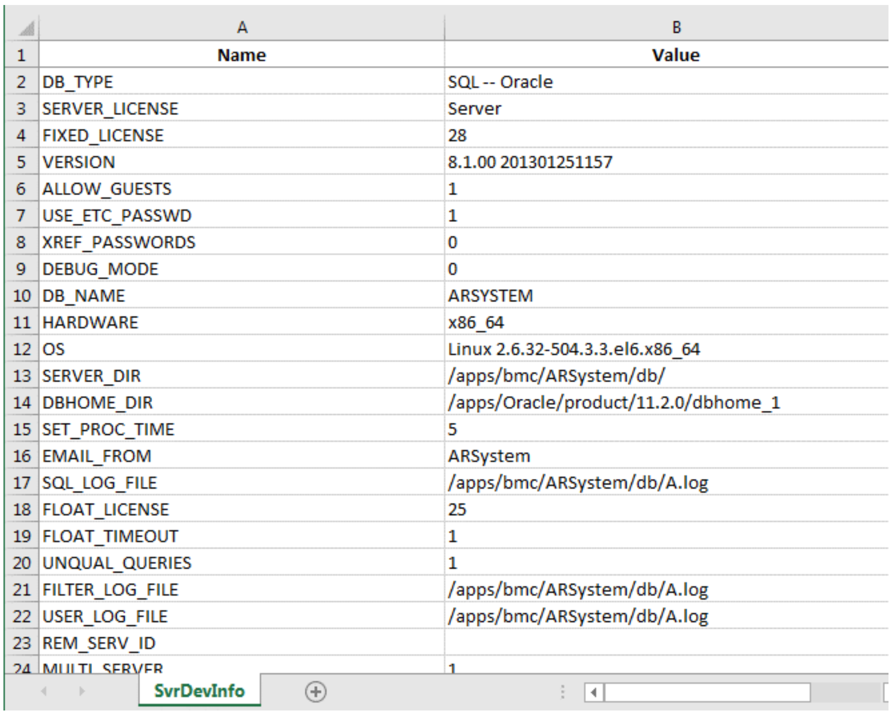
Usage Instructions

```
Function:
.   This Meta-Update script makes a CSV from all the automatic
fields and values in the the automatic Tag:  AR_INFO
.     Output CSV file in the form:
.        Name              Value
.       DB_TYPE           SQL -- SQL Server
.       VERSION           7.6.04 Build 002 201101141059
.       ALLOW_GUESTS      1
.       DB_NAME           ARSystem
.
. Usage
.   SthMupd   000-SvrInfo  Do  -outf   out-file
.     where       -outf        is the output CSV file name
                            (overwritten)
.
. Examples
.   SthMupd   000-SvrInfo  Do  -outf   devsvrinfo.csv
```

Sample Output

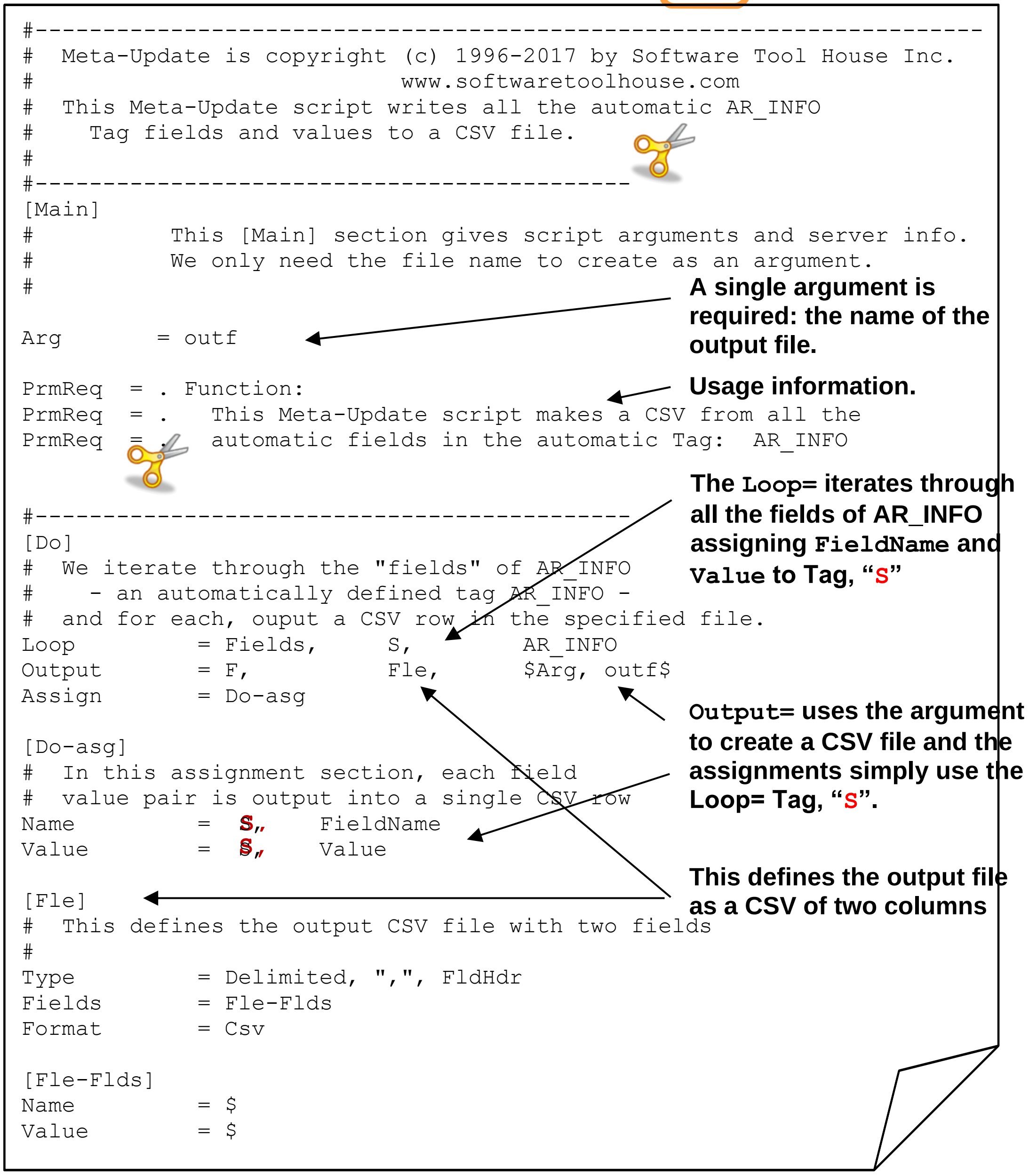
```
>> SthMupd.exe 000-SvrInfo.ini Do -outf SvrDevInfo.csv
Meta-Update    Version 5.74 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2017 by Software Tool House Inc.
               www.softwaretoolhouse.com
i FoDfInit: Opened file SvrDevInfo.csv for Output= of
            000-SvrInfo.ini [Fle] line: 59.
i [Do] Lp:  1 of 347: Fld: AR_INFO, DB_TYPE
i [Do] Lp:  2 of 347: Fld: AR_INFO, SERVER_LICENSE
i [Do] Lp:  3 of 347: Fld: AR_INFO, FIXED_LICENSE
i [Do] Lp:  4 of 347: Fld: AR_INFO, VERSION
i [Do] Lp:  5 of 347: Fld: AR_INFO, ALLOW_GUESTS
i [Do] Lp:  6 of 347: Fld: AR_INFO, USE_ETC_PASSWD
i [Do] Lp:  7 of 347: Fld: AR_INFO, XREF_PASSWORDS
i [Do] Lp:  8 of 347: Fld: AR_INFO, DEBUG_MODE
```

```
i [Do] Lp:  346 of 347: Fld: AR_INFO, MAX_LOG_HISTORY
i [Do] Lp:  347 of 347: Fld: AR_INFO, SUPRESS_LOGOFF_SIGNALS i
[Do] Lp:  eof 347 record OK; 0 records with errors; total: 347.
i Statistics:
i          Sections:                     1
i          Maximum section depth:        1
i          Loops:                        1
i          Loop values:                347   errors:      0
i terminating successfully in 2 sec.
```



	A	B
1	**Name**	**Value**
2	DB_TYPE	SQL -- Oracle
3	SERVER_LICENSE	Server
4	FIXED_LICENSE	28
5	VERSION	8.1.00 201301251157
6	ALLOW_GUESTS	1
7	USE_ETC_PASSWD	1
8	XREF_PASSWORDS	0
9	DEBUG_MODE	0
10	DB_NAME	ARSYSTEM
11	HARDWARE	x86_64
12	OS	Linux 2.6.32-504.3.3.el6.x86_64
13	SERVER_DIR	/apps/bmc/ARSystem/db/
14	DBHOME_DIR	/apps/Oracle/product/11.2.0/dbhome_1
15	SET_PROC_TIME	5
16	EMAIL_FROM	ARSystem
17	SQL_LOG_FILE	/apps/bmc/ARSystem/db/A.log
18	FLOAT_LICENSE	25
19	FLOAT_TIMEOUT	1
20	UNQUAL_QUERIES	1
21	FILTER_LOG_FILE	/apps/bmc/ARSystem/db/A.log
22	USER_LOG_FILE	/apps/bmc/ARSystem/db/A.log
23	REM_SERV_ID	
24	MULTI_SERVER	1

Development time:
under fifteen minutes!



```
#---------------------------------------------------------------------
#   Meta-Update is copyright (c) 1996-2017 by Software Tool House Inc.
#                          www.softwaretoolhouse.com
#   This Meta-Update script writes all the automatic AR_INFO
#     Tag fields and values to a CSV file.
#
#--------------------------------------------
[Main]
#           This [Main] section gives script arguments and server info.
#           We only need the file name to create as an argument.
#

Arg        = outf

PrmReq   = . Function:
PrmReq   = .   This Meta-Update script makes a CSV from all the
PrmReq   = .   automatic fields in the automatic Tag:  AR_INFO


#--------------------------------------------
[Do]
#   We iterate through the "fields" of AR_INFO
#     - an automatically defined tag AR_INFO -
#   and for each, ouput a CSV row in the specified file.
Loop           = Fields,     S,       AR_INFO
Output         = F,          Fle,     $Arg, outf$
Assign         = Do-asg

[Do-asg]
#   In this assignment section, each field
#   value pair is output into a single CSV row
Name           =  S,     FieldName
Value          =  S,     Value

[Fle]
#   This defines the output CSV file with two fields
#
Type           = Delimited, ",", FldHdr
Fields         = Fle-Flds
Format         = Csv

[Fle-Flds]
Name           = $
Value          = $
```

A single argument is required: the name of the output file.

Usage information.

The `Loop=` iterates through all the fields of AR_INFO assigning `FieldName` and `Value` to Tag, “S”

`Output=` uses the argument to create a CSV file and the assignments simply use the Loop= Tag, “S”.

This defines the output file as a CSV of two columns

005-ArSchema

This beginner's script creates a CSV of the tables in an ARS server with additional columns for and the number of records they contain.

It does a **`QuerySql=`** on the **`arschema`** table with an **`Output=`** in the same section. The **`Output=`** column assignments also use **`QuerySql=`** to get counts.

The script demonstrates:

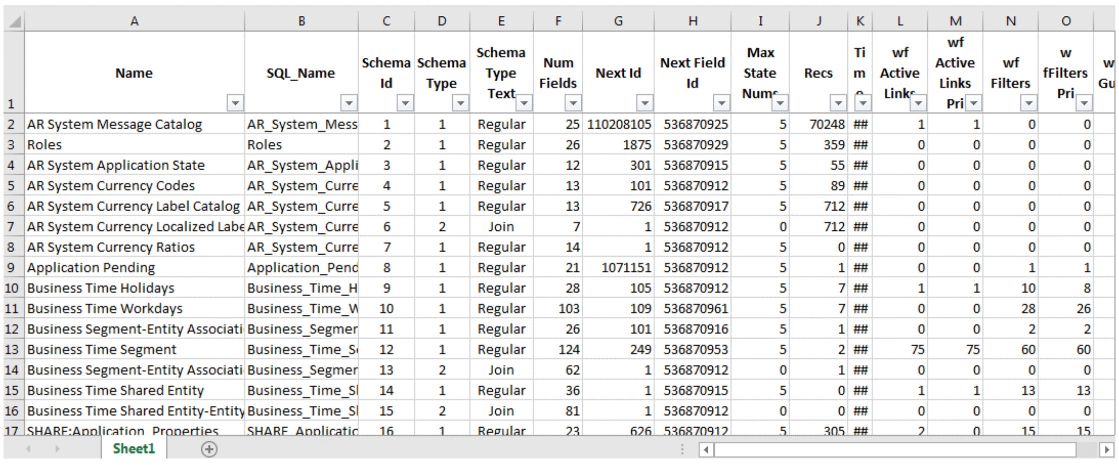
- How to use a **`QuerySql=`** statement.
- How to use an **`Output=`** to create a CSV
- How to use **`QuerySql=`** in **`LookUp`** assignments

Usage Instructions

```
. Function:
.    Produces a report of tables, number of records,and various
.      workflow counts from arschema
.
. Usage:  005-ArSchema  Do -fout  output_file_name  -ptn "ptn"
.
.   where fout      is the output file name
.         ptn       if entered, selects only some table names
.                   Default "%"
.
. Note:   You may set an alternate CSV separator with the
environment
.           variable:  SthCsvSep.
.         For example, to set a semi-colon:
.
.           set SthCsvSep=;
.
. Examples
.    005-ArSchema.ini  Do -fout   ArSchRpt-all.csv
.    005-ArSchema.ini  Do -fout   ArSchRpt-CI.csv
                       -ptn   "BMC.CORE:%"
.
```

Sample Output

```
>> SthMupd.exe 005-ArSchema.ini Do -outf SvrDevInfo.csv
Meta-Update    Version 5.74 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2017 by Software Tool House Inc.
               www.softwaretoolhouse.com
i [Do] One:Opened file cent-arschema.csv for Output=
           in 005-ArSchema.ini [F-out] line: 238.
i   [DoV] Sql: 1 of 3849: PDL:SLIInterface_Create,457
i   [DoV] Sql: 2 of 3849: PDL:SoftwareLibraryItem,458
i   [DoV] Sql: 3 of 3849: PDL:SoftwareLibraryItemSearch,459
i   [DoV] Sql: eof 3849 record OK; 0 records with errors;
i [Do] One: 1 record OK; 0 records with errors; total: 1.
i Statistics:
i         Sections:                   2
i         Maximum section depth:      2
i         SQL queries:                1
i         SQL records:             3849   errors:      0
```



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
1	Name	SQL_Name	Schema Id	Schema Type	Schema Type Text	Num Fields	Next Id	Next Field Id	Max State Nums	Recs	Ti m	wf Active Links	wf Active Links Pri	wf Filters	w fFilters Pri	w Gu
2	AR System Message Catalog	AR_System_Mess	1	1	Regular	25	110208105	536870925	5	70248	##	1	1	0	0	
3	Roles	Roles	2	1	Regular	26	1875	536870929	5	359	##	0	0	0	0	
4	AR System Application State	AR_System_Appli	3	1	Regular	12	301	536870915	5	55	##	0	0	0	0	
5	AR System Currency Codes	AR_System_Curre	4	1	Regular	13	101	536870912	5	89	##	0	0	0	0	
6	AR System Currency Label Catalog	AR_System_Curre	5	1	Regular	13	726	536870917	5	712	##	0	0	0	0	
7	AR System Currency Localized Labe	AR_System_Curre	6	2	Join	7	1	536870912	0	712	##	0	0	0	0	
8	AR System Currency Ratios	AR_System_Curre	7	1	Regular	14	1	536870912	5	0	##	0	0	0	0	
9	Application Pending	Application_Pend	8	1	Regular	21	1071151	536870912	5	1	##	0	0	1	1	
10	Business Time Holidays	Business_Time_H	9	1	Regular	28	105	536870912	5	7	##	1	1	10	8	
11	Business Time Workdays	Business_Time_W	10	1	Regular	103	109	536870961	5	7	##	0	0	28	26	
12	Business Segment-Entity Associati	Business_Segmer	11	1	Regular	26	101	536870916	5	1	##	0	0	2	2	
13	Business Time Segment	Business_Time_S	12	1	Regular	124	249	536870953	5	2	##	75	75	60	60	
14	Business Segment-Entity Associati	Business_Segmer	13	2	Join	62	1	536870912	0	1	##	0	0	0	0	
15	Business Time Shared Entity	Business_Time_Sl	14	1	Regular	36	1	536870915	5	0	##	1	1	13	13	
16	Business Time Shared Entity-Entity	Business_Time_Sl	15	2	Join	81	1	536870912	0	0	##	0	0	0	0	
17	SHARE:Application_Properties	SHARE_Applicatic	16	1	Regular	23	626	536870912	5	305	##	2	0	15	15	

Sheet1

Software Tool House Inc

Development time:
under fifteen minutes!

```
#   Meta-Update is copyright 1996-2017 by Software Tool House Inc.
#   File:                     005-ArSchema.ini
#                              Meta-Update sample script.
#                              Shows QuerySql=, Output=, QuerySql= in
#                                   LookUps, regex pattern splitting
#--------------------------------------------
[Main]
#   The main section gives sign-on info and declares
#     script arguments required and usage info.$

Arg           = fout
Arg           = Ptn                Default "%"

PrmReq        = 1, . Function:
PrmReq        = .    Produces a CSV from arschema of
PrmReq        = .      tables, counts of records and workflow

#--------------------------------------------
[Do]
#   We simply do a QuerySql= and an Output=
#    with an AssignInit to set the QuerySql text
AssignInit    = asg-Init
QuerySql      = Tbl, ArSchV,                                           &
                  select                                               &
                    name, schemaid, schematype,                        &
                    nextfieldid, nextfieldid, maxstate..               &
                    viewname,  timestamp                               &
                  from arschema                                        &
                    $Ptn, Qual$
Output        = F, F-out, $Arg, fout$
Assign        = asg-CsvRow

[asg-Init]
#   A where clause to ”” or “where name like ‘$Arg, Ptn$’”
@Cmd         = Ref, Ptn,   al,   ""
@Cmd         = @if("$Arg, Ptn$" != "")                                 &
               Ref, Ptn, Qual,   "where name like '$Arg, Ptn$'"


#[asg-CsvRow]
# if the table is a Join or Regular, we will
#   assign a record count via an SQL LookUp
Name                = Tbl, name
SQL_Name            = V,   name_sql
SchemaId            = Tbl, 02
SchemaType          = Tbl, schematype
SchemaTypeText      = Tbl, schematype
NumFields           = Tbl, 04
NextId              = Tbl, nextid
```

Usage Instructions

The `AssignInit` sets a `where` clause for the `QuerySql=`

Script entry point. Issues a `QuerySql` for all regular forms in `arschema` possibly with a `where` clause.

`QuerySql` results can be referenced by number, like BMC Remedy, or, as a set of fields with interpretations applied as in `[ArSchV]`

`Output=` in the same section that iterates, adds one row each iteration.

The `Output=` assignments reference the SQL results by position or name.

`SchemaTypeText` has an interpretation in the **`OupPut=`** file definition: **`[F-out]`**

```
NextFieldId        = Tbl, 06
MaxStateNums       = Tbl, 07
Records            = @if("$Tbl, schematype$" == 1 ||             &
                         "$Tbl, schematype$" == 2         )      &
                     @LookUp, GetRecCount,                       &
                              $Tbl, viewname$
Time               = Tbl, time


wfActiveLinks      = @LookUp,  LkpAL,       $Tbl, sch
wfActiveLinksPri   = @LookUp,  LkpALp,      $Tbl, sch
wfFilters          = @LookUp,  LkpF,        $Tbl, schemaid$
```

Only on Reguar and Join statements will this `LookUp` and `QuerySql=` be done.

The time field is interpreted by the field declaration to be a Remedy timespamp field.

These `LookUps` do a `QuerySql` that returns a `select count(*)`

```
#--------------------------------------------------
[F-out]
Type          = Delimited, "$Cfg, CsvSep$", FldHdr
Format        = Excel
Field         = F-out-Fld

[F-out-Fld]
# These are the CSV file's fields
#
Name              = $
SchemaId          = $
SchemaType        = $
SchemaTypeText    = $        Subst /0/Null/                      &
                             Subst /1/Regular/                   &
                             Subst /2/Join/                      &
                             Subst /3/View/                      &
                             Subst /4/Dialog/                    &
                             Subst /5/Vendor/
#--------------------------------------------------
[ArSchV]
name             = $
schemaid         = $
schematype       = $
time             = $           Date: epoch
```

`[F-out]` defines the output file – columns and automatic transformations

After assignments but before output and values in this column will have these character substitutions applied.

`QuerySql` results are interpreted into fields be a Field section such as `[ArSchV]`

```
#--------------------------------------------------
#- Work-flow count lookups
#  All given schema id (not name!)
[LkpAL]
QuerySql         = qAL,                                          &
                   @na,                                          &
                   select count(*) from actlink_mapping          &
                   where  schemaId = $CTL, LookUp_Src$
QuerySqlTarget = $qAL,   1$

[LkpALp]
QuerySql         = qALp,                                         &
                   @na,                                          &
                   select count(*) from actlink_mapping          &
                   where  schemaId = $CTL, LookUp_Src$           &
                   and    objindex = 0
QuerySqlTarget = $qALp,  1$
```

Each `LookUps` is called with a `SchemaId` and does a `QuerySql=` returning one row and one column: a `select count(*)`

600-ItsmVer

This simple script outputs a message with the version of ITSM running on the server.

The section has a single assignment section and no iteration at all. That assignment section assigns the ITSM version through a **`QuerySql= LookUp`** on: SHARE:Application_Properties

A single argument is needed to prevent the Usage information display.

The script demonstrates:

- How to use a simple **`AssignInit`** in a useful script..
- How to use an **`QuerySql=`** to assign a value

Usage Instructions

```
Function:
. This is a Meta-Update script that reports the ITSM version
.
. Usage
.   SthMupd   600-ItsmVer  Do  -go
.     where       -go          is required but ignored
.
. Examples
.   SthMupd   600-ItsmVer  Do  -go
..
```

Sample Output

```
>> SthMupd.exe 600-ItsmVer.ini  Do  -go
Meta-Update    Version 5.74 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2017 by Software Tool House Inc.
               www.softwaretoolhouse.com
i FoDfInit: Opened file SvrDevInfo.csv for Output= of
            000-SvrInfo.ini [Fle] line: 59.
i [Do] Msg: .
i [Do] Msg: .
i [Do] Msg: ItsmVer: 8.1.00
i [Do] Msg: .
i [Do] Msg: .
i [Do] One:
i [Do] One: 1 record OK; 0 records with errors; total: 1.
i Statistics:
i         Sections:                    1
i         Maximum section depth:       1
i         Singleton Sections:          1   errors:      0
i terminating successfully in 1 sec.
```

Development time:
under fifteen minutes!

```
#   Meta-Update is copyright 1996-2017 by Software Tool House Inc.
#   File:                    005-ArSchema.ini
#                             Meta-Update sample script.
#                             Shows QuerySql=, Output=, QuerySql= in
#                                  LookUps, regex pattern splitting
#---------------------------------------------
[Main]
#  The main section gives sign-on info and declares
#    script arguments required and usage info.

Arg        = go

PrmReq     = . Function:
PrmReq     = .   This Meta-Update script reports
PrmReq     = .     the ITSM version
PrmReq     = .

#---------------------------------------------
 [Do]
#  This has only an Initial Assignment section
AssignInit    = Do-asgInit

[Do-asgInit]
#  In this assignment section, we use a LookUp to get the version
#    and then display it
@Cmd                  = Ref,  V,  ItsmVer,
                        @LookUp,
                        Lkp-Version,   -dmy-
@Cmd                  = Msg,  I,  .
@Cmd                  = Msg,  I,  .
@Cmd                  = Msg,  I,  ItsmVer: $V, ItsmVer$
@Cmd                  = Msg,  I,  .
@Cmd                  = Msg,  I,  .

[Lkp-Version]
QuerySql         = Qver, @na,                                         &
                   select property_value                              &
                   from   SHARE_Application_Properties                &
                   where  Property_Name    = 'Version'       and      &
                          Application_GUID = (                        &
                     select Application_GUID                          &
                     from   SHARE_Application_Properties              &
                     where  Property_Name    = 'Name'        and      &
                            Property_Value   = 'BMC Atrium CMDB'      &
                                             )
QuerySqlTarget = Qver,  1$
```

Usage Instructions

The `AssignInit` does all the work. It uses a LookUp to get the ITSM Version and then issues a message with it.

Sets `$V, ItsmVer$` through a `QuerySql= LookUp`.

`$V, ItsmVer$` is simply used in our message.

The `LookUp` uses a `QuerySql=` to select a single row in `SHARE:Application Properties`

610-ItsmAppProp

This simple script makes a CSV of SHARE:Application_Properties filling in the Display Only Application Name column.

The script demonstrates:

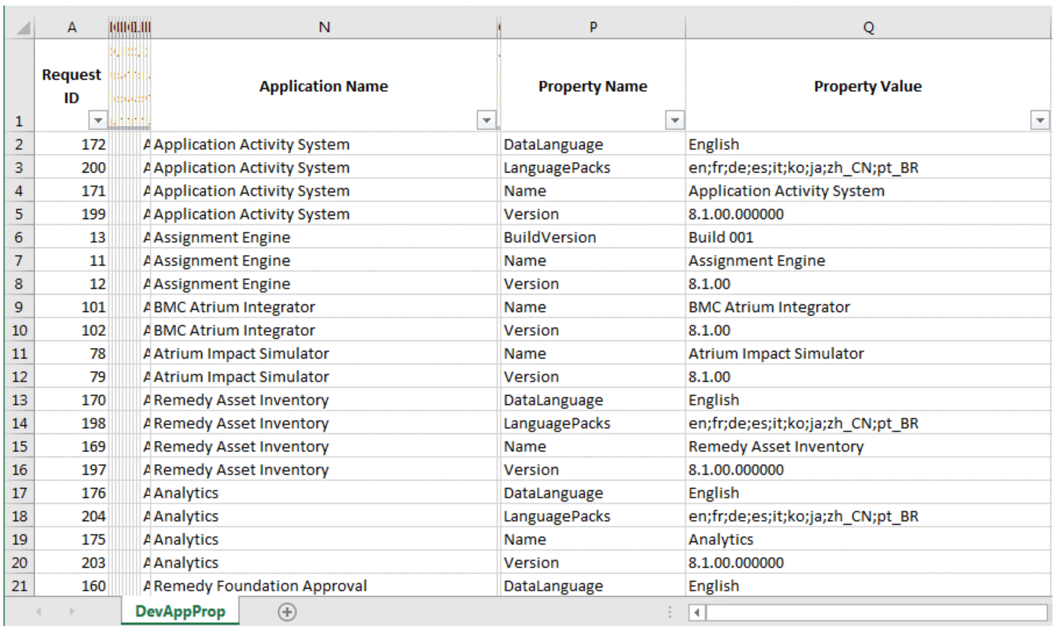
- How to use a simple **`AssignInit`** in a useful script..
- How to use an **`QuerySql=`** to assign a value

Usage Instructions

```
      Function:
.   This script makes a CSV of SHARE:Application_Properties
.   effecting a LookUp to add the App Name column.
.
. Usage
.   SthMupd   610-ItsmAppProp  Do  --outf   csv-file
.     where       outf        is the output CSV file
.
. Examples
.   SthMupd   610-ItsmAppProp  Do  -outf   DevAppProp.csv
.
```

Sample Output

```
>> SthMupd.exe 610-ItsmAppPropr.ini  Do  -outf DevAppProp.csv
Meta-Update    Version 5.74 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2017 by Software Tool House Inc.
               www.softwaretoolhouse.com
i FoDfInit: Opened file DevAppProp.csv for Output= of
            610-ItsmAppProp.ini [Fle-Application_Properties]
            line: 71.
Qry: 1 of 305: DataLanguage       5  English
Qry: 2 of 305: LanguagePacks      5
en;fr;de;es;it;ko;ja;zh_CN;
Qry: 3 of 305: Name               5  Application Activity
System
Qry: 4 of 305: Version            5  8.1.00.000000

[Do] Qry: 304 of 305: Name        5  Task Management System
[Do] Qry: 305 of 305: Version     5  8.1.00.000000
[Do] Qry: 305 of 305: 305 record OK; 0 records with errors;
total: 305.
Statistics:
        Sections:                    1
        Maximum section depth:       1
        Queries:                     1
        Query records:             305   errors:      0
i terminating successfully in 4 sec.
```



	Request ID	Application Name	Property Name	Property Value
2	172	Application Activity System	DataLanguage	English
3	200	Application Activity System	LanguagePacks	en;fr;de;es;it;ko;ja;zh_CN;pt_BR
4	171	Application Activity System	Name	Application Activity System
5	199	Application Activity System	Version	8.1.00.000000
6	13	Assignment Engine	BuildVersion	Build 001
7	11	Assignment Engine	Name	Assignment Engine
8	12	Assignment Engine	Version	8.1.00
9	101	BMC Atrium Integrator	Name	BMC Atrium Integrator
10	102	BMC Atrium Integrator	Version	8.1.00
11	78	Atrium Impact Simulator	Name	Atrium Impact Simulator
12	79	Atrium Impact Simulator	Version	8.1.00
13	170	Remedy Asset Inventory	DataLanguage	English
14	198	Remedy Asset Inventory	LanguagePacks	en;fr;de;es;it;ko;ja;zh_CN;pt_BR
15	169	Remedy Asset Inventory	Name	Remedy Asset Inventory
16	197	Remedy Asset Inventory	Version	8.1.00.000000
17	176	Analytics	DataLanguage	English
18	204	Analytics	LanguagePacks	en;fr;de;es;it;ko;ja;zh_CN;pt_BR
19	175	Analytics	Name	Analytics
20	203	Analytics	Version	8.1.00.000000
21	160	Remedy Foundation Approval	DataLanguage	English

Development time:
under fifteen minutes!

```
#    Meta-Update is copyright 1996-2017 by Software Tool House Inc.
#    File:                    610-ItsmAppProp.ini
#                             Meta-Update sample script.
#                             Shows Query=, Output=, Copying fields, LookUps
#--------------------------------------------
[Main]

Arg        = go

PrmReq     = . Function:
PrmReq     = .   This Meta-Update script makes a CSV of
PrmReq     = .     SHARE:Application_Properties
PrmReq     = .

#--------------------------------------------
[Do]
Query         = Src,                                                   &
                SHARE:Application_Properties,                          &
                @sort(Application GUID, Property Name)                 &
                1=1
Output        = Fle,                                                   &
                Fle-Application_Properties,                            &
                $Arg, outf$
Assign        = Do-asg                                                 &


[Do-asg]
#
#   In this output CSV assignment, we copy the
SHARE:Application_Properties record
#
Application Name   = @LookUp,    Lkp-AppName,                          &
                     $Src, Application GUID$
@Cmd          = Copy,   Src

#--------------------------------------------
[Lkp-AppName]
Default       = "Error:  Property Name not found for $CTL, LookUp_Src$
Cache         = 0
NoMatch       = W, Default
QuerySql      = AppName,          @na,                                 &
                select   property_value                                &
                from     SHARE_Application_Properties                  &
                where    Application_GUID = '$CTL, LookUp_Src$'  and   &
                         Property_Name     = 'Name'
QuerySqlTarget = $AppName,1$

[Fle-Application_Properties]
#
#   This "File Section" declares the output CSV file
Type          = Delimited, ",", FldHdr
Format        = Csv
Fields        = Fle-Application_Properties-Fields

[Fle-Application_Properties-Fields]
@Cmd           = Copy,  SHARE:Application_Properties
```

Usage Instructions.
The Query= processes all records in the table.

Because the `Output=` follows the Query=, one row of the CSV is written for each Query= row returned.

We set the Display Only field, with a `QuerySql=` `LookUp`.

Assignments are pretty simple. The CSV has the same named fields as the form, we just copy them.

The `LookUp` uses a `QuerySql=` to select a single row in `SHARE:Application Properties`

Using a `Cache=` saves queries.

The Copy command copies all fields from a schema.

900-SwLogs

This sample can be used to control server logging. Use it to set all log files and turn logging on and off..

The script demonstrates:

- How to use a simple **Update=** to set log files by writing to a vendor form introduced in ARS 7.1
- How to set a special Tag, **AR_INFO**, **DEBUG_MODE** to control the server.

Usage Instructions

```
      Function:
 .   This Meta-Update script switches the ARserver log files and
.     sets logging on or off by assigning DEBUG_MORE in AR_INFO
.
. Usage
.   SthMupd   900-SwLogs.ini  Do  -off    -log
.   SthMupd   900-SwLogs ini  Do  -log    log_file
.                                 -dbg    DebugModeValue
.
.   where   -off    sets DEBUG_MODE to 0 (off);
                    does NOT change log files
.                   Note: -log is a required arg but is ignored
.           -log    is a log file name without a path and extension
.                   Note: path, ".log" are configurable in the script
.           -dbg    a specific DEBUG_MODE value;
                    the default is configured in the script
. Examples
.                   >> Turn logging off:
.   SthMupd   900-SwLogs.ini Do -off
.                   >> Turn logging on and set log files to:
.                   >>    "/apps/bmc/ARSystem/db/my.log"
.   SthMupd   900-SwLogs.ini  Do -log   my
.                   >> Set all log files as above & turns logging off
.   SthMupd   900-SwLogs.ini. Do  -log   my   -dbg 0
..
```

Sample Output

```
>> SthMupd.exe  220-SwLogs.ini Do -off -log
Meta-Update    Version 5.74 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2017 by Software Tool House Inc.
               www.softwaretoolhouse.com
i [Do] One:
i [Do] One: Launching:  1 of  1 [DoOn] from @if(! "$Arg, off$")
i   [DoOn] One: Updated AR System Administration:
                         Server Information, Id: 000000000000001
i Statistics:

i         Output Schema records:       1   updated
i         Outputs OK:                  1
i         Outputs Errors:              0
i terminating successfully in 1 sec.
```

Development time:
under 30 minutes!

```
#   Meta-Update is copyright 1996-2018 by Software Tool House Inc.
#   File:                 900-SwLogs.ini
#   Function:             Set Server Logging and switch log files
#
[Main]
Arg        = off           Default   0
Arg        = log
Arg        = dbg           Default   ""

PrmReq     = . Function:
PrmReq     = .   script switches the ARserver log
PrmReq     = .     files and sets logging on by
PrmReq     = .

#---------------------------------------------
[Do]
#      It does no Queries and so does a single record update to
#        a hard coded request id
#
AssignInit    = asg-Cfg
AssignInit    = Do-asgInit
Launch        = @if(! "$Arg, off$")    DoOn

[DoOn]
Update        = PIOtst,                                              &
                AR System Administration: Server Information,        &
                '1' = "000000000000001"
Assign        = Do-asg
AssignTerm    = Do-asgTerm

[Do-asgInit]
# turn debug_mmode Off unless already Off
@Cmd          = @if("$AR_INFO, DEBUG_MODE$")                          &
                Ref,    AR_INFO,      DEBUG_MODE,      0

[Do-asgTerm]
# set the DEBUG_MODE to turn tracing on now
# make a debug_mode mask
@Cmd          = @if("$Arg, dbg$")
  @Cmd          = Ref,    AR_INFO,      DEBUG_MODE,      $Arg, dbg$
@Cmd          = else
  @Cmd          = Ref,    AR_INFO,      DEBUG_MODE,      $Cfg,
Dbg_Default$
@Cmd          = endif

[Do-asg]
#      First, set full path passed log file
@Cmd       = Ref,   X,   @na,     @regex,                               &
             #(.*)[\\\\/](.*)#,    $Arg,   log$
@Cmd       = @if("$X, @rc$")
  @Cmd       = Ref,  V,   LogNm,    $Arg, log$
@Cmd       = else
  @Cmd       = Ref,  V,   LogNm,    "$Cfg, LogPth$$Arg, log$$Cfg, LogSfx$"
@Cmd       = endif
apilogfile            = V, LogNm
filterlogfile         = V, LogNm
sqllogfile            = V, LogNm
```

Usage Instructions.

The `AssignInit` turns *off* logging.

We don't do any more if `off` was specified.

We update the only record in the vendor form added since 7.1.

The assignments set the log files.

The `AssignTerm` turns *on* logging once the log file names are set.

We adjust the passed log file name by prepending a configured directory and suffixing a configured extension unless the passed argument included directory slashes.

910-SvrInfo-set

This one line sample can be used to set ARS Server INFO parameters such as logging, Admin Mode, Mid-Tier passwords.

The script demonstrates:

- How to use a simple **`AssignInit=`** to set a single **`AR_INFO`** value.

Usage Instructions

```
      Function:
.   Used to change dynamic server settings such as admin mode,
.     logging, Midtier passwords, and so on.
.
.   Writes to a single AR_INFO key with the supplied
.     value to the current ARS server: ?????
.
.   Run script 000-SvrInfo.ini to get current keys and values.
.
. Usage:
.   SthMupd.exe 910-SvrInfo-set Do -key key  -val val
.
. where:
.   -key         is a writable "Field Name" from the AR_INFO tag.
.   -val         is the new value that the key can accept
.
. Notes:
.   Specifying non-writable key, or non-acceptable values cause
.     script errors with no effects.  For example, specifying
.    -val "apple" for a -key DEBUG_MODE (-val must be an integer).
.   Specifying acceptable but invalid values can cause errors in
.     the running ARS Server.  For example, specifying
.    -val "/nodir/server_trace.log" will be accepted but fail later.

. Warnings:
.   ./conf/ar.conf or .\confar.cfg is NOT updated!
.
. Examples
.   SthMupd   910-SvrInfo-set  Do  -key API_LOG_FILE
                                   -val "/nodir/server_trace.log"
.             sets a log file and turns on logging;
.             log file failure if there is no directory /nodir.
.   SthMupd   910-SvrInfo-set  Do  -key MID_TIER_PASSWD
.                                  -val "arsystem"
.   SthMupd   910-SvrInfo-set  Do  -key APP_SERVICE_PASSWD
.                                  -val "arsystem"
```

Sample Output

```
>> SthMupd.exe  910-SvrInfo-set.ini Do -key DEBUG_MODE -val 0
Meta-Update    Version 5.74 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2017 by Software Tool House Inc.
               www.softwaretoolhouse.com
i [Do] One:
i [Do] One: 1 record OK; 0 records with errors; total: 1.
i Statistics:
i         Sections:                    1
```

```
i          Maximum section depth:        1
i          Singleton Sections:           1    errors:      0
i terminating successfully in 1 sec.
```

Development time:
under five minutes!

```
#   Meta-Update is copyright 1996-2017 by Software Tool House Inc.
#   File:                 910-SvrInfo-set.ini
#                         Meta-Update sample script.
#                         Shows a simple assignment
#--------------------------------------------
[Main]

Arg        = key
Arg        = val

PrmReq     = . Function:
PrmReq     = .   Changes dynamic server settings
PrmReq     = .     such as admin mode, logging,
PrmReq     =       Midtier passwords, and so on.

#--------------------------------------------
[Do]
AssignInit    = Do-asg

[Do-asg]
@Cmd          =  Ref,  AR INFO  $Arg, key$,   $Arg, val$
```

Two required arguments.

Usage Instructions.

The `AssignIni` does all the work – one assignment.

`AR_INFO` is a special tag. When you assign to it, the equivalent server info is set on the server.

460-Change-Approve

This samples moves ITSM Changes in Scheduled for Approval status to the next state by Approving them.

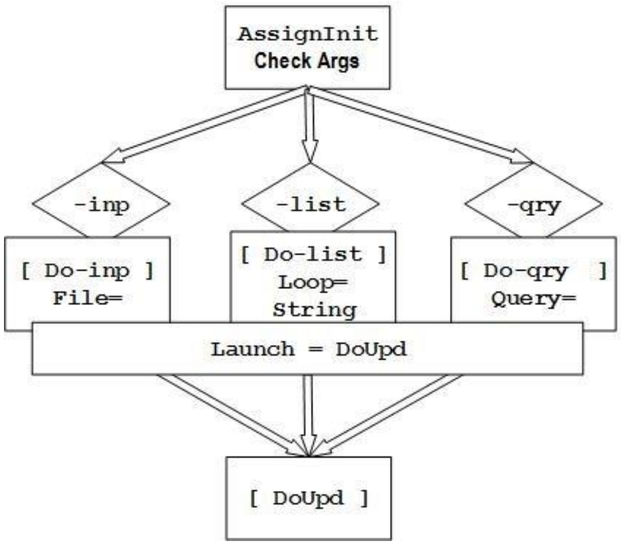
It takes three different inputs:

- A comma separated list of Infrastructure Change ID
- A CSV file with a column called Infrastructure Change ID
- Any query on CHG:Infrastructure Change

This script processes the input, ensuring Changes are in Scheduled for Approval status, approving the changes, and optionally, moving them to their next phase.

The script demonstrates:

- how to make a script operate on different inputs and yet use the same process.
- a File=, Loop=, or Query= are used to select the Changes that are in Status: Scheduled for Approval.
- How to throw an error if a selected Change is not in the correct Status.
- The script now calls a single section that adds or updates a signatire record.



- Then, it updates a Signature-Change Join record to validate the process.

Usage Instructions

`Function:`

```
. Function:
.   We move Changes in Scheduled For Approval status
.     through writes to AP:Signature (Override approver)
.
. Updates:   AP:Signature
.
. Usage:
.                 One of three forms to select Changes:
.             *** use only one of -inp, -qry, or, -list ***
.
.   SthMupd   460-Change-Approve.ini   Do   -
.                    inp      file_of_change_ids                   -go
.   SthMupd   460-Change-Approve.ini   Do
.                   -qry      query on CHG:Infrastructure Change   -go
.   SthMupd   460-Change-Approve.ini   Do
.                   -list     list of Change IDs                   -go
. Warning:
.   The argument -NextStage 1 will update the Change to move it
.   from Scheduled to Implementation In Progress.  This is NOT
.   recommended as Merge will be used to avoid group permissions.
.   Audit logs, etc, will not be create
.
. where
.   -inp    change_file       A CSV file with a column called
.                               Infrastricture Change ID on row 1
.   -qry    query text        A Query on CHG:Infrastructure Change
.       -start nn    -max nn    with -qry a batching of records.
.                                    default: 0, 0 (all)
.   -list   Change_IDs        A comma separated list of
.                               Infrastructure Change IDs
.
. Examples
.   SthMupd   460-Change-Approve.ini   Do   -go   -inp  change.scsv
.   SthMupd   460-Change-Approve.ini   Do   -go
.            -qry     "'6' > \"04/12/2016\" and '6' < \"04/11/2016\""
.   SthMupd   460-Change-Approve.ini   Do    -go
.             -qry     "'1' = \"CRQ000001000017\" or
.                       '1' < \"CRQ000001000012\""
.   SthMupd   460-Change-Approve.ini   Do    -go
.             -list    "CRQ_CAL_1000011,CRQ_CAL_1000006"
.
```

Sample Output

```
>> SthMupd.exe  460-Change-Approve.ini Do
                -go -list CRQ000000000119 -NextStage
Meta-Update    Version 5.74 (x64) for ARS lib 9.1.0
           (c) Copyright 1996-2017 by Software Tool House Inc.
               www.softwaretoolhouse.com
i [Do] One:
i [Do] One: Launching:  1 of  3 [Do-list]
            from @if("$V, sec$" == "list")Do-list
i   [Do-list] Lp:  1 of 1: Str: CRQ000000000119
i   [Do-list] Lp:  1 of 1: Launching:  1 of  1 [DoUpd]
                           from @if("$V, Do$")DoUpd
i     [DoUpd] One:
i     [DoUpd] One: Updated   schema: AP:Detail-Signature,
                             Id:
000000000000349|000000000000433
```

```
i     [DoUpd] One: Launching:  1 of  1 [DoUpd-Chg]
                   from @if("$Arg, NextStage$")DoUpd-Chg
i       [DoUpd-Chg] Qry: 1 of 1: CRQ000000000119nullMupd
null
                                Ben Chernynulltest change
i       [DoUpd-Chg] Qry: 1 of 1: Merge OK- op:merge
                    Schema = CHG:Infrastructure Change
                    ID = CRQ000000000212 Old ID =
CRQ000000000212
i       [DoUpd-Chg] Qry: 1 of 1: 1 record OK; 0 records with
errors
i     [DoUpd] One: 1 record OK; 0 records with errors
i   [Do-list] Lp:  eof 1 record OK; 0 records with errors;
total: 1.
i [Do] One: 1 record OK; 0 records with errors; total: 1.
i Statistics:
i         Sections:                   4
i         Maximum section depth:      4
i         Assignment Sections:        2
i         Singleton Sections:         2   errors:      0
i         Queries:                    1
i         Query records:              1   errors:      0
i         Loops:                      1
i         Loop values:                1   errors:      0
i         Output Schema records:      2   updated  (with 0
skipped)
i         Outputs OK:                 2
i terminating successfully in 8 sec.
```

Software Tool House Inc

Development time:
under two hours!

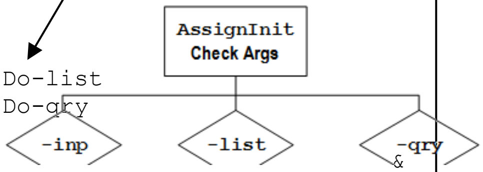
```
#   Meta-Update is copyright 1996-2018 by Software Tool House Inc.
#   File:           460-Change-Approve.ini
#  Function:        We process a query (| list|file) of changes in status
#                   Scheduled For Approval and approve those changes
#---------------------------------------------
[Main]
Arg          = go
Arg          = qry                 Default  ""
Arg          = inp                 Default  ""
Arg          = list                Default  ""
Arg          = start               Default  0
Arg          = max                 Default  0
Arg          = NextStage           Default  0

PrmReq  = . Function:
PrmReq   = .  We process a list or query of Change
PrmReq   = .  in cheduled For Approval approving

#---------------------------------------------
[Do]
AssignInit     = Do-asgInit
Launch         = @if("$V, sec$" == "list") Do-list
Launch         = @if("$V, sec$" == "qry" ) Do-qry
Launch         = @if("$V, sec$" == "inp" )
IdLog          = IdLog                                             &
                 On          All,                                  &
                 Fdef        Fout-IdLog,                           &
                 Fname       $CTL, ScriptFx$-$CTL, Pid$-idlog.csv, &
                 Fasg        Fout-IdLog-asg

[Do-asgInit]
@Cmd           = Ref,  V,    Err,                 ""
@Cmd           = Ref,  V,    Msg,                 ""
@Cmd           = Ref,  V,    Do,                  0
@Cmd           = Ref,  V,    sec,                 ""
@Cmd           = @if("$Arg, qry$" && ! ("$Arg, list$" || "$Arg, inp$"))
  @Cmd           = Ref,  V,    sec,                 qry
@Cmd           = endif
@Cmd           = @if("$Arg, list$" && ! ("$Arg, qry$" || "$Arg, inp$"))
  @Cmd           = Ref,  V,    sec,                 list
@Cmd           = endif
@Cmd           = @if("$Arg, inp$" && ! ("$Arg, qry$" || "$Arg, list$"))
  @Cmd           = Ref,  V,    sec,                 inp
@Cmd           = endif
@Cmd           = @if(! "$ V, sec$")
  @Cmd           = Abort,    E,    ...Please specify one of        &
                   -inp, -list, or -qry
@Cmd           = endif
```

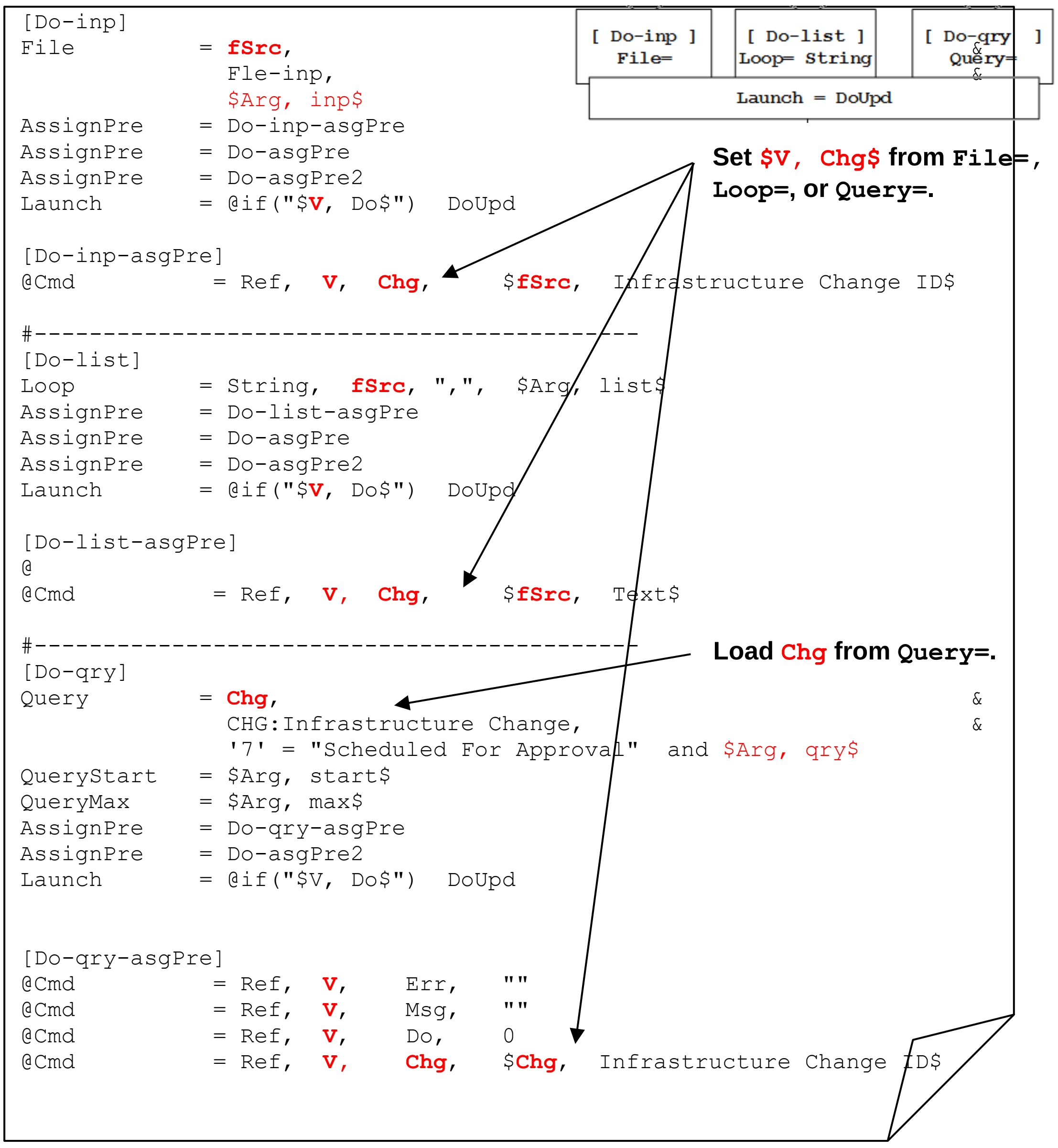
Arguments are checked in `AssignInit=`

Usage Instructions.

We want an `IdLog` CSV created.

Only one section is Launched.





```
[Do-inp]
File          = fSrc,
                Fle-inp,
                $Arg, inp$
AssignPre     = Do-inp-asgPre
AssignPre     = Do-asgPre
AssignPre     = Do-asgPre2
Launch        = @if("$V, Do$")  DoUpd

[Do-inp-asgPre]
@Cmd            = Ref,   V,   Chg,      $fSrc,   Infrastructure Change ID$

#--------------------------------------------
[Do-list]
Loop          = String,  fSrc,  ",",   $Arg, list$
AssignPre     = Do-list-asgPre
AssignPre     = Do-asgPre
AssignPre     = Do-asgPre2
Launch        = @if("$V, Do$")  DoUpd

[Do-list-asgPre]
@
@Cmd            = Ref,   V,   Chg,      $fSrc,   Text$

#--------------------------------------------
[Do-qry]
Query         = Chg,                                                  &
                CHG:Infrastructure Change,                            &
                '7' = "Scheduled For Approval"  and $Arg, qry$
QueryStart    = $Arg, start$
QueryMax      = $Arg, max$
AssignPre     = Do-qry-asgPre
AssignPre     = Do-asgPre2
Launch        = @if("$V, Do$")  DoUpd


[Do-qry-asgPre]
@Cmd            = Ref,   V,     Err,    ""
@Cmd            = Ref,   V,     Msg,    ""
@Cmd            = Ref,   V,     Do,     0
@Cmd            = Ref,   V,     Chg,    $Chg,  Infrastructure Change ID$
```

```
# This is used by 2 of 3 above sections as a common asgPre to pick
# up the change into the tag Chg from
# $V, Chg$ - loaded from a file or list
[Do-asgPre]
@Cmd             = Ref,  V,   Err,      ""
@Cmd             = Ref,  V,   Msg,      ""
@Cmd             = Ref,  V,   Do,       0
@Cmd             = Ref,  V,   gotChg,          @LookUp,                         &
                   Lkp-Chg,   $V,  Chg$
@Cmd             = @if(! "$V, gotChg$")
  @Cmd             = Ref,  V,  Err,    Change not found.
  @Cmd             = Ref,  V,  Msg,    Change $V, Chg$ not found.
  @Cmd             = Abort,  E,   $V,  Err$ - $V, Msg$
@Cmd             = else
  @Cmd             = @if("$Chg, Change Request Status$" !=                     &
                             "Scheduled For Approval")
    @Cmd             = Ref,  V,  Err,    Change in wrong Status
    @Cmd             = Ref,  V,  Msg,    Change $V, Chg$ not                   &
                       inScheduled For Approval ($Chg, 7$)
    @Cmd             = Abort,  E,   $V,  Err$ - $V, Msg$
  @Cmd             = endif
@Cmd             = endif


[Lkp-Chg]
# Loads a CHG:Infrastructure Change into CHG
Default          = ""
NoMatch          = D, Default
Query            = Chg,                                                        &
                   CHG:Infrastructure Change,                                  &
                   'Infrastructure Change ID'  =  "$CTL, LookUp_Src$"
QueryTarget      = $Chg, 1$
```

Load Chg from **Query=**. in **LookUp for File=** and **Loop=.**

Throw errors is change was not found or is in the wrong status.

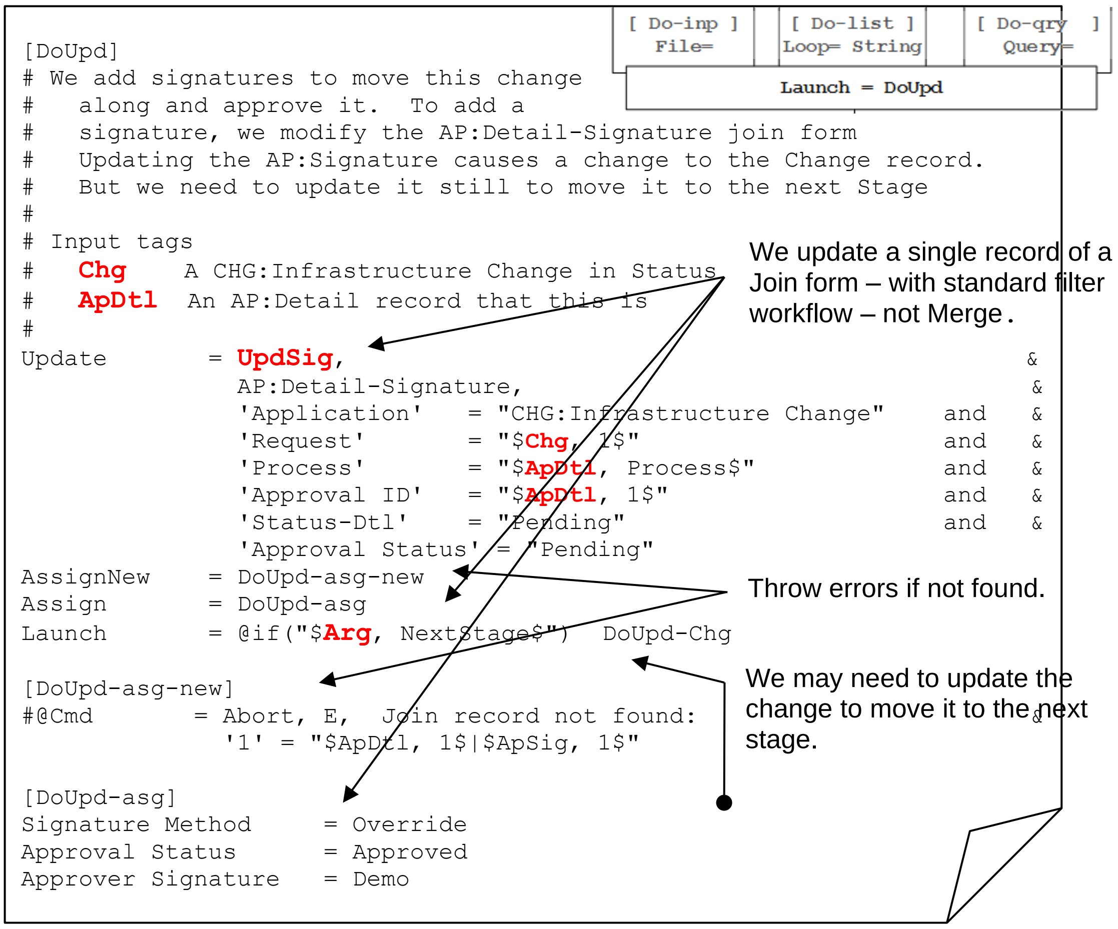
```
[Do-asgPre2]
#
# Used by all 3 above sections as a common asgPre
# we have a loaded CHG:Infrastructure Change in Chg
#
#  We need to load a two more records here
#    1) AP:Detail    in ApDtl
#    2) AP:Signature in ApSig
#
@Cmd          = @if(! "$V, Err$")
  @Cmd          = Ref,  V,   gotApDtl,    @LookUp,
                  Lkp-ApDtl,     $V,  Chg$
  @Cmd          = @if(! "$V, gotApDtl$")
    @Cmd          = Ref,  V,  Err,   AP:Detail not found
    @Cmd          = Ref,  V,  Msg,  .Change $V, Chg$'s                &
                                 AP:Detail not found.
    @Cmd          = Abort,  E,  $V, Err$ - $V, Msg$
  @Cmd          = else
    @Cmd          = Ref,  V,   gotApSig,                              &
                    @LookUp,                                          &
                    Lkp-ApSig,     $V,  Chg$
    @Cmd          = @if(! "$V, gotApSig$")
      @Cmd          = Ref,  V,  Err,   AP:Signature not found
      @Cmd          = Ref,  V,  Msg,  .Change $V, Chg$'s              &
                      AP:Signature for AP:Detail $ApDtl, 1$ not found.
      @Cmd          = Abort,  E,   $V, Err$ - $V, Msg$
    @Cmd          = else
      @Cmd          = Ref,  V,   Do,    1
    @Cmd          = endif
  @Cmd          = endif
@Cmd          = endif

[Lkp-ApDtl]
# Uses "Chg" - a Change in Waiting For Authorization to pick up the
#   single AP:Detail record that will need to be signed.
Default       = ""
NoMatch       = D, Default
Query         = ApDtl,                                                &
                AP:Detail,                                            &
                'Application'     = "CHG:Infrastructure Change"   and &
                'Request'         = "$Chg, 1$"                    and &
                'Process'         = "$Chg, ApprovalProcessName$"
QueryTarget   = $ApDtl, 1$

[Lkp-ApSig]
# Uses "Chg" - a change record, and ApDtl, an AP:Detail record to
# pick up the single AP:Signature record that will need to be signed.
Default       = ""
NoMatch       = D, Default
Query         = ApSig,                                                &
                AP:Signature,                                         &
                'Approval Status'    = "Pending"                  and &
                'Approval ID'        = "$ApDtl, 1$"
QueryTarget   = $ApSig, 1$
```

Load ApDtl from **Query=** in **LookUp** using data from Chg.

Throw errors if not found.

Load ApSig from **Query=** in **LookUp** using data from Chg and ApDtl.

Throw errors if not found.

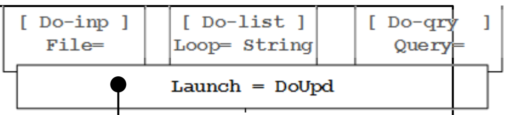


```
[DoUpd]
# We add signatures to move this change
#   along and approve it.  To add a
#   signature, we modify the AP:Detail-Signature join form
#   Updating the AP:Signature causes a change to the Change record.
#   But we need to update it still to move it to the next Stage
#
# Input tags
#   Chg     A CHG:Infrastructure Change in Status
#   ApDtl   An AP:Detail record that this is
#
Update        = UpdSig,                                              &
                AP:Detail-Signature,                                  &
                'Application'    = "CHG:Infrastructure Change"    and &
                'Request'        = "$Chg, 1$"                     and &
                'Process'        = "$ApDtl, Process$"             and &
                'Approval ID'    = "$ApDtl, 1$"                   and &
                'Status-Dtl'     = "Pending"                      and &
                'Approval Status' = "Pending"
AssignNew     = DoUpd-asg-new
Assign        = DoUpd-asg
Launch        = @if("$Arg, NextStage$")  DoUpd-Chg

[DoUpd-asg-new]
#@Cmd        = Abort, E,  Join record not found:                       &
               '1' = "$ApDtl, 1$|$ApSig, 1$"

[DoUpd-asg]
Signature Method      = Override
Approval Status       = Approved
Approver Signature    = Demo
```

We update a single record of a Join form – with standard filter workflow – not Merge.

Throw errors if not found.

We may need to update the change to move it to the next stage.



```
#
[DoUpd-Chg]
# The Change has been updated to the Scheduled status
#   but we also want move the stage.
# We need to wait as the Signature update causes
#   a Change Status update, but with a delay.  For now, a hard
#   coded 5 secs using the gnu (sygwin) sleep (on path).
# Finally, our user is unlikely to belong to the right group to
#   work on the change, so we will move it along by faking the
#   effects of the workflow (perhaps missing audit logs etc)
#
AssignInit    = DoUpd-Chg-asgInit
Update        = ChgUpd,                                   &
                CHG:Infrastructure Change,                &
               '179' = "$Chg, 179$"
Merge         = Yes, NoWorkflow
Assign        = DoUpdChg-asg

[DoUpd-Chg-asgInit]
@Cmd                    = Spawn,    sleep 5s

[DoUpdChg-asg]
@Cmd        = @if("$ChgUpd, Change Request Status$" != "Scheduled")
  @Cmd        = Ref,  V,   Err,  Update failed; Change is in wrong Status
  @Cmd        = Ref,  V,   Msg,   Change $V, Chg$ not                &
                Scheduled ($ChgUpd, 7$); Is delay (5s) enough?)
  @Cmd        = Abort,  E,   $V, Err$ - $V,Msg$
@Cmd        = endif

Change Request Status          = Implementation In Progress
CurrentStageNumber             = 4
ChangeRequestStatusString      = Implementation In Progress
Change Request Prev Status     = Scheduled


#---------------------------------------------
# Do's IdLog file, our Err, Msg, standard stuff, and
#   some fields from the record
#
[Fout-IdLog]
Type                              = Delimited, ",", FldHdr
Format                            = Csv
Fields                            = Fout-IdLog-fields

[Fout-IdLog-fields]
Err                               = $
Msg                               = $

[Fout-IdLog-asg]
Err                               = V,      Err
Msg                               = V,      Msg
```

We need to update the change to move it to the next stage.

We update the same Change record, this time using **Merge** and inhibiting workflow.

Wait 5 seconds so Remedy CAI can effect the Change.

Throw an error if Change in wrong Status.

Make assignments needed.

IdLog= File, Fields, Assignments

Ticket Creation Batch Command

This is an invented script built as an example to help learn Meta-Update. The script is untested and it must be noted that the script will need editing before being run in any reader's environment.

Requirements

We need a simple, easy to use, parameterized, ticket generator for our ARS Help Desk. We want to be able to create new tickets so that we can, if desired, force an assignment to a specific group.

We want to use this callable command in various ways:

- ➢ Remedy ARS workflow and escalations,
- ➢ Scheduled jobs through "at" or "cron",
- ➢ Configured commands in other their network monitors
- ➢ Added as a last step of some of their bespoke software

The command would depend on the arguments given. Defaults would be assumed for all null arguments.

❖ Requester Email or Requester login If it contained an "@" it would be looked up in a people form as an email. Otherwise it would be looked up as a login name.	
❖ Subject	The subject of the ticket.
❖ Description	The full textual description.
❖ Category	If not supplied, use "Default"
❖ Type	
❖ Item	
❖ Assignment Group	Only assign if supplied.

Meta-Update solution

Development time: ***one*** **hour!**

```
[Main]
Server          =      Sth2
User            =      Demo
ArgNm           =      Subject
ArgNm           =      ReqSearch
ArgNm           =      Description
ArgNm           =      Category
ArgNm           =      Type
ArgNm           =      Item
ArgNm           =      AsgGrp

PrmReq          =      2

[TT-New]
#Simply create a Ticket every time.
Schema              =   HPD:HelpDesk
Assign              =   Asg-New-TT

[Asg-New-TT]
Subject         =   Arg, Subject
Description     =   Arg, Description

@Cmd                =   @if(“$Arg, ReqSearch$ ==””)
  LoadQ             =   Req,                                      &
                        SHR:People,                               &
                        ‘Login’    =   “Default Requester”)
@Cmd                =   else
  @Cmd              =   @if(“$Arg, ReqSearch$ ~=”@”)
    LoadQ           =   Req,                                      &
                        SHR:People,                               &
                        ‘Email’    =   “Default Requester”)
  @Cmd              =   else
    LoadQ           =   Req,                                      &
                        SHR:People,                               &
                        ‘Login’    =   “Default Requester”)
  @Cmd              =   endif
@Cmd                =   endif

Requester Id        =   Req, 1
Requester Login     =   Req, Login

Category    =   @if(“$Arg, Category$ == “”,                       &
                    “Default”,                                    &
                    “$Arg, Category$”)
Type        =   @if(“$Arg, Type$ == “”,                           &
                    “Default”,                                    &
                    “$Arg, Type$”)
Item        =   @if(“$Arg, Item$ == “”,                           &
                    “Default”,                                    &
                    “$Arg, Item$”)

Assignment Group =  @if(“$Arg, AsgGrp$” != “”)                    &
                    Arg, AsgGrp
```

Names the arguments

Specifies that only 2 arguments are required and usage info when not enough arguments supplied.

Simple command section that always creates a single record in the HelpDesk schema

Simple assignment of passed argument value

Load the requester record into memory under the tag, *Req*

Assignment of data from loaded Requester record.

Assign either “Default” or the supplied values.

Only make this assignment if a value was supplied.

The Category, Type, and Item assignments are simply based on the passed arguments on an individual basis. To make similar assignments on a hierarchical basis, simply use this segment instead or the three individual Category, Type, Item assignments above:

```
@Cmd       =  @if("$Arg, Category$ == "")
  Category    =  "Default"
@Cmd          =  else
  Category    =  Arg, Category
  @Cmd        =  @if("$Arg, Type$ == "")
    Type      =  "Default"
  @Cmd        =  else
    Type      =  Arg, Type
    @Cmd      =  @if("$Arg, Item$ == "")
      Item    =  "Item"
     @Cmd     =  else
      Item    =  Arg, Item
     @Cmd     =  endif
  @Cmd     =  endif
@Cmd       =  endif
```

The PrmReq can be used to specify usage information as well as the required number of arguments. The usage information is delivered when an insufficient number of arguments is supplied on the command line. Note that passing a null value – "" – is still passing a value. Named arguments not supplied on the command line contain the null value.

This example is equivalent to the above but will supply usage information when used incorrectly.

```
PrmReq     =  4, Usage:
PrmReq     =  .   TT-New -p  Subj, Desc, Req,  Cat, Typ, Item, AsgGrp
PrmReq     =  .  where
PrmReq     =        Subj      is required and is the ticket short subject
PrmReq     =        Desc      is required and is the long
PrmReq     =        Req       is either the requester login or email
address
PrmReq     =                  Default Requester assumed if null
PrmReq     =        Cat       Category (Default if null)
PrmReq     =        Typ       Type     (Default if null)
PrmReq     =        Itm       Item     (Default if null)
PrmReq     =        AsgGrp    is an assignment group or null
PrmReq     =  .  Create a ticket and optionally assigns it to a group
```

Closed Ticket Replicator

This is taken from a customer solution. It has been modified to be used as a Meta-Update sample. The script demonstrates how to launch other dependent command sections, how to make assignments from multiple records, how to use the Copy assignment command.

Background

The customer had a series of Perl scripts to control ticket generation and filing emails with tickets. This allowed a full email conversation between the ticket agent and ARS system and the requester.

Sometimes a requester would reply to an email after it was closed. The customer's business process stated no further work could be done on a closed ticket.

As such, a mechanism would be needed to create a new ticket from the old ticket selecting work history records and emails.

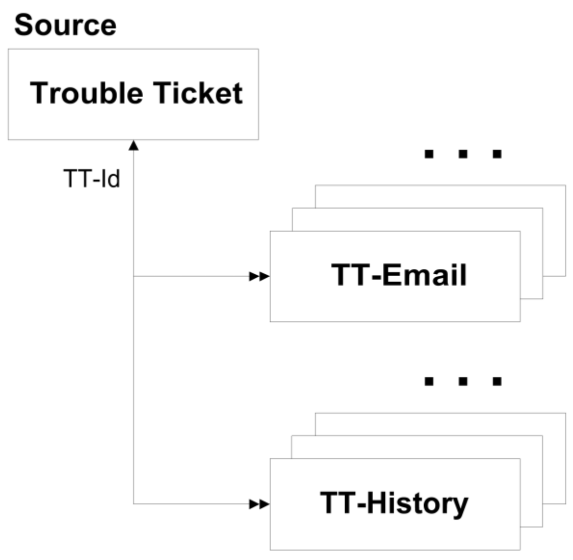
Requirements

A Perl callable ticket replicator was needed. It would create a new, open, assigned ticket, containing the emails, the work history with a few extra generated records identifying the email to the closed ticket. It would copy pertinent data from the old ticket.

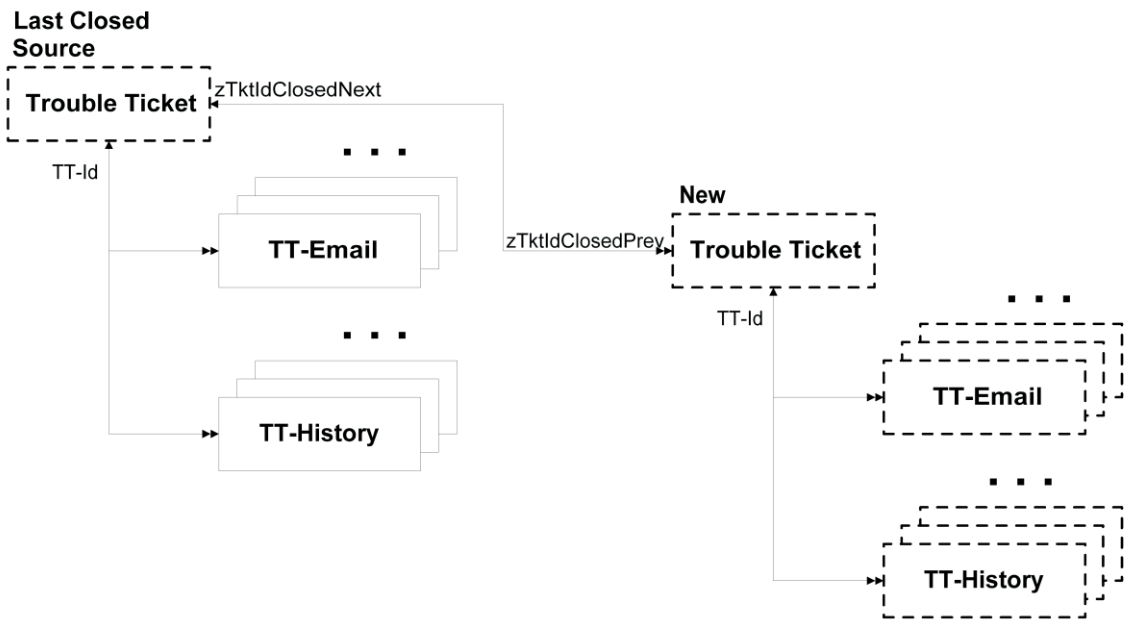
The new ticket would be created assigned to the resolving group of the closed ticket.

The two tickets would be linked for a GUI facility to allow ticket chains to be followed. The closed ticket would need to be updated with the new ticket's id.

This image shows the schemas and records of a single ticket.



The dashed lines in this image show the desired updated and created records:



Meta-Update solution

Development time: *three* hours!

```
[Main]
Server          =       Sth2
User            =       Demo
ArgNm           =       TtIdClosed
ArgNm           =       IdLog
PrmReq          =       2

[TT-Copy]
# this section creates a Ticket every time.
Schema                =   TT-TroubleTicket
LoadQ                 =   Src_TT,
                          TT-TroubleTicket,
                          '1' = "$Arg, TT-IdClosed$"
Create                =   New_TT,
                          TT-TroubleTicket
Merge                 =   Yes
Assign                =   asg-TT-New
Launch                =   TT-Orig-Upd,
Launch                =   TT-Email
Launch                =   TT-Hist-1, TT-Hist-2,
                          TT-Hist-3, TT-Hist-4,
                          TT-Hist-5
Launch                =   TT-New-Upd

[asg-TT-New]

Status                 = New
zTktIdClosed           = Src_TT, 1
zTktIdClosedNew        = $NULL$
@Cmd                   = @if("$Src_TT, Next Action$" != "")
  Next Action            = "Old closed ticket actions:\n"
  Next Action            = "==========================\n"
@Cmd                   = endif
Next Action            = Src_TT, Next Action
Ticket Type            = Problem
Priority               = Medium
Severity               = 4
Ticket Opened          = $DATE$
Ticket Closed          = $NULL$
Problem Started        = $NULL$
Problem Fixed          = $NULL$
Escalate when          = $NULL$
# The next cmd copies all non-assigned fields.
@Cmd                   = Copy, Src_TT, DupIgnore, CoreAssign, Skip: 1
```

Names two required arguments.

The closed source TT is loaded into memory from the passed Id.

Always creates one single record in the HelpDesk schema

Merge API is used to inhibit Submit filters.

The created record is loaded into memory after submission and other sections are run to copy dependent records.

Assignments for the new TT some arbitrary values (constants) and the remaining set of fields from the closed ticket

Meta-Update solution

The closed source ticket and the newly created, open ticket are in memory before these sections are called.

This section links the new TT on the old one using the Merge API.

```
[TT-Orig-Upd]
# Update the original closed TT with the
#   Newly Opened TT ID
Query               =   UpdOrig_TT,                                          &
                        TT-TroubleTicket,                                    &
                        '1' = "$Arg, TT-IdClosed$"
Merge               =   Yes
Update              =   UpdOrig_TT
Assign              =   TT-Orig-Upd-1

[TT-Orig-Upd-1]
zTktIdClosedNew       = New_TT, 1
Action Log            = "Email received after closure; New TT created: "
Action Log            = New_TT, 1
Action Log            = "\n"
```

This section copies all the source emails to the newly created ticket. This is a copy of records in a single form. Merge is used to prevent notifications.

```
[TT-Email]
Schema          = TT-Email
Query           = Src_TT-E, TT-Email,                                        &
                       'Ticket-ID' = "$Src_TT, 1$"
Update          = Upd_TT-E, TT-Email,                                        &
                       'Ticket-ID' = "$New_TT, 1$"      AND                  &
                       'Date Sent' = "$Src_TT-E, Date Sent$"
Assign          = TT-EmailUpd
Update0         = TT-EmailUpd
Merge           = AllowNull, SkipPatternMatch


Ticket-ID       = New_TT,    1
@Cmd            = Copy, Src_TT-E, DupIgnore, CoreAssign
```

The newly created ticket id is assigned and all remaining fields from the old email are copied.

The Main section

The `PrmReq`= specifies that three arguments are required. A better one might be:

```
PrmReq  = 3, TT-Closed-Copy.ini copies a closed TT to a new, unassigned TT
PrmReq  = .
PrmReq  = .  usage
PrmReq  = .    SthMupd.exe  TT-Closed-Copy.ini TT-Copy  -p  TT-ID-src  IdLog
PrmReq  = .
PrmReq  = .  where
PrmReq  = .    TT-ID-src     Parm 1    the closed ticket's ID that will be copied
PrmReq  = .    IdLog         Parm 2    the file name for the IdLog
PrmReq  = .
PrmReq  = .  function
PrmReq  = .    Will create a new TT as a copy of the old one including all its
PrmReq  = .      previous emails but not its history records for which a few will
PrmReq  = .      be artificially generated.
PrmReq  = .    Will also update the source TT with the newly generated ID plus
PrmReq  = .      a text reference to the generation...
PrmReq  = .    The source TT must not have already been copied to a new TT.
PrmReq  = .
```

TT-Copy, The Called Command Section.

The command section called to copy a ticket is: `TT-Copy`.

To call the command, either on the command line or within a shell script or batch file, one could enter:

```
    SthMupd.exe  ./TT-Cpy.mus  TT-Copy  -p   TKT000049  TKT000049  /tmp/..
```

`TT-Copy` has no `Query=`, `QuerySql=`, `File=` so it is executed exactly once.

The `Load=` keyword loads the closed source ticket. The data of this ticket can be referenced with `$Src_TT, field$`. This can be used in subsequent queries or assignments.

The `Create=` keyword causes an ARS record to be submitted. This could have been an Update= keyword which would have allowed different assignments for an update or a create operation. An ARS query that selects exactly one or zero update records must be specified.

It loads the source ticket record which is always the last ticket closed in a chain of tickets. That id is passed on the command line as the named argument, `TT-Closed`.

After the command section creates the new ticket, that new ticket is re-read so that all fields have the current values, and the launches are processed in order.

Launching Other Command Sections.

Each launch allows a new command section to be processed. That command process has all the preceding sections' references available to it. It can query and iterate like any other section.

```
    Launch           =   TT-Orig-Upd,
    Launch           =   TT-Email
    Launch           =   TT-Hist-1, TT-Hist-2,                          &
                         TT-Hist-3, TT-Hist-4, TT-Hist-5
    Launch           =   TT-New-Upd
```

Command Section	Overview

`TT-Copy`	The called or main section. It executes only once and creates a new ticket. It then launches, in order, these other sections.
`    TT-Orig-Upd`	Uses a Merge to add the new ticket id reference to the old ticket.
`    TT-Email`	Uses a Query= to copy all emails to the new ticket.
`    TT-Hist`	Uses a Query= to copy all the history records.
`    TT-Hist-1, 2, ..5`	Uses a Create= to create a few new history records for the TT-Closed-Copy operation.
`    TT-New-Upd`	

Server Delta Copy

This script is created as a learning vehicle to demonstrate several Meta-Update statements.

Requirements

A reporting server must be kept in sync with a production server. The sync job is run on a 24 hour delay basis. The updated records are to be transferred based on the last modification date. Request IDs are to be maintained. The subset of the tables to be kept synchronised is given by an ASCII file. That file also specifies query text that can be appended to the programmed modification date query.

The following is a sample file

```
Tbl,TblSql,IdFld,ModFld,QueryText

SHR:People,shr_people,request_id,modified_on

HPD:HelpDesk,hpd_helpdesk,case_id_,modified_on

SHR:Audit,shr_audit,request_id,
     'Schema 1' = \"HPD:HelpDesk\"

SHR:Association,shr_association,request_id,
     modified_on,'Schema 1' = \"HPD:HelpDesk\"
```

Names five file columns.

The fifth value is null.

Appended to programmed query, isolates the Help Desk associated records for a run with this file.

Interestingly, multiple jobs can be simultaneously to take advantage of the ARS server's multi-threading. This could be extended to several machines. Each job would specify independent sets of dependent tables.

Script Overview

The Main section will define the source server. It will also change the date into a format suitable for an SQL query.

The called command section will process the passed CSV file. It will not make any outputs itself, but instead, launch another command section.

That launched section, will in turn issue an SQL Query on the table named in the CSV and a date with any optional query text appended.

That query section will actually do an SQL query to prevent ARS timeouts as generally the modified by field is not indexed. It will iterate through that list updating any records it needs to.

This Script Demonstrates

- Processing a CSV with a File=.
- Using an assignment section to prepare a query string.
- Using an assignment section to convert a date from a normal format to an integer for an SQL query.
- Using a Read Server. In a LoadQ and a QuerySql.
- Specifying an Update query.
- Using the Copy assignment command.
- Using a Launch.

Meta-Update script

``

```
[Main]
Server          =       Dev01
User            =       Demo
ReadServers     =       Main-Prod
ArgNm           =       inp-csv-fle
ArgNm           =       mod-date
AArgNm          =       idlog
PrmReq          =       3
IdLog           =       $Arg, idlog$.log

[Main-Prod]
Tag             =       Prod
Server          =       Dev02-prod-copy
User            =       Demo
Port            =       3201

[Fle-Tbl]
Type            =       Delimited, “,”,FldHdr
Format          =       Excel
Fields          =       Fle-Tbls-Flds

[Fle-Tbl-Flds]
Tbl             =       $       # table name in ARS
TblSql          =       $       # table name in SQL view
IdFld           =       $       # ‘1’ in SQL
ModFld          =       $       # ‘8’ in SQL
QueryText       =       $       # SQL query text

[SvrSync-Date]
# Processes the passed CSV file of tables to synchronise.
File                =   Ftbls,                                          &
                        Fle-Tbl,          ,                             &
                        $Arg, inp-csv-fle$”
AssignPre           =   asg-Mk-Qry
Launch              =   Tbl-Sync

[asg-Mk-Qry]
# will append an “and” and any extra query text
#   supplied in the CSV row
@Cmd = Ref, Vars, Qry
       $Ftbls, ModFld$ > $Arg, mod-date$
@Cmd = @if(“$Ftbls, QueryText$” != “”)                              &
       Ref, Vars, Qry   $Vars,  Qry$ AND ( $Ftbls, QueryText$ )
```

Specifies the script’s tag, ip and login for the production server.

Names three arguments.

All arguments are required.

Declares the format and field name for the passed CSV file.

The file’s first record contains the field names which must match these fields.

This is the called section. It iterates through the file’s rows.

The `AssignPre=` section is run after the next file record is loaded but before any Launches are processed.

This makes an “and ..” string if the CSV had an optional QueryText value.

```
[Tbl-Sync]
# Issues an SQL query to obtain the modified
#   record IDs, Loads the records and updates
#   them on the target server.

QuerySql      =   @Prod,                                          &
                  SqlLst,                                         &
                  @na,                                            &
                  select $Ftbls, IdField$                         &
                  from   $Ftbls, TblSql $                         &
                  where  $Vars, Qry$

LoadQ         =   @Prod,                                          &
                  Src,                                            &
                  $Ftbls, Tbl$,                                   &
                  '1'=   "$SqlLst, 1$"

Update        =   Tgt,                                            &
                  $Ftbls, Tbl$,                                   &
                  '1'=   "$SqlLst, 1$"
Merge         =   Yes, NoWorkflow
Assign        =   asg-Copy
AssignNew     =   asg-Copy

[asg-Copy]
@Cmd          = Copy, Src, CoreAssign
```

This section has the CSV row loaded and does the rest of the work by issuing the SQL Query on the source server for the modified request ids, loading the record, and updating the record on the target server.

This section copies the source record's fields including core fields.

Script Detail

The `[Main]` section does these things:

1. Specifies three argument names with the `ArgNm=` keyword.
2. Specifies the file to be generated as the id log with the `IdLog=` keyword..
3. Says that all three arguments are required but does not give additional user help text when those arguments are not specified on the command line.
4. Establishes the server and authentication parameters for the update server
5. Establish the server and authentication parameters to the source server through the `ReadServers=` keyword. The value of that keyword is a section name which, like the Main section gives server and authentication parameters for addition servers. Note the `Tag=` keyword in the `[Main-Prod]` section. Queries will use this tag - `@Prod` - to reference the addition server.

The Called Command Section

The `[SvrSync-Date]` section is specified on the command line and is the script "entry-point".

The `File=` keyword says we will iterate through a columnar file. The `[Fle-Tbl]` section specifies the attributes and fields of the file. Row one of the file contains the field names and must match the fields specified in the CSV.

The `AssignPre=` allows us to build the select SQL query for the modified date using the fields as specified in the file row and the optional query text also specified in the file row.

The first assignment of `[asg-Mk-Qry]` makes the modification date query text for the SQL statement using the modification field name specified in the CSV file for this table and the time argument passed on the command line. This is set in tag “Vars”, field “Qry”.

If the CSV query text was non-null, the same string is appended with “and (..)” using the supplied query text.

Now that the SQL query string has been made, the section launches the actual worker section `[Tbl-Sync]` to copy the modified records. This section has no output.

The Launched Section

Section `[Tbl-Sync]` is launched once for each table / row in the passed configuration file row. That row is in memory when this launched section is invoked. In addition, a select Query string has been created.

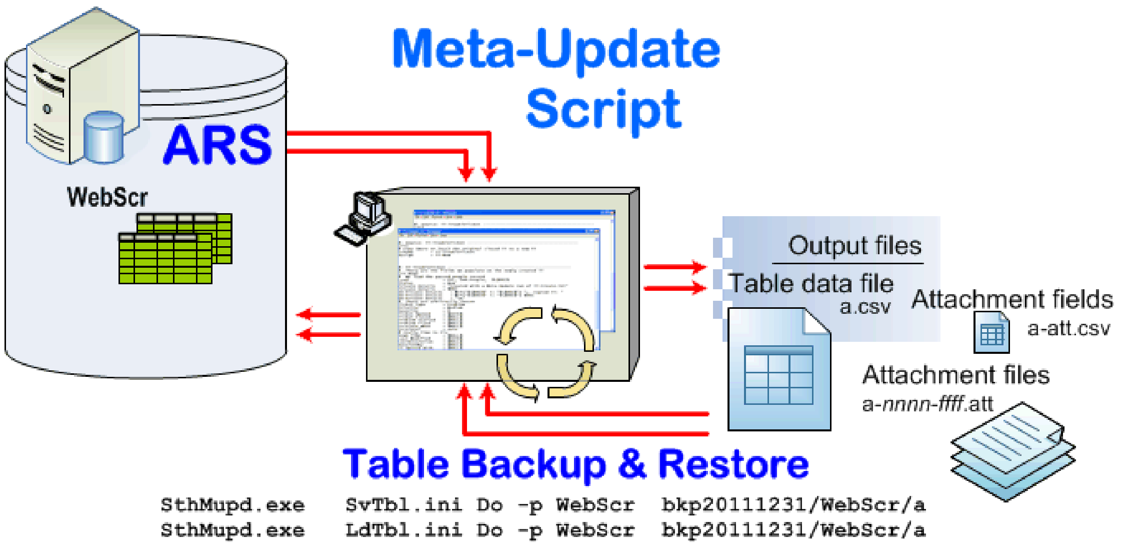
This section issues a select to retrieve the ids of the modified records for the given table. It does this with the `QuerySql=` keyword, specifying the @Prod server tag. The @na says that we will not name or edit any of the columns returned by the select statement, instead referring to them by their column numbers.

We iterate through the set of Request Ids returned by the select. During each iteration, we load the source record from the source server with the `LoadQ=` keyword, and issue the `Update=` to create the same record on the target server with the same request id as in the source server. That Update or Create is performed using the Merge API and no filters are fired – including filters set to fire or Megre.

The `Assign=` and `AssignNew=` sections are the same and simply issue the Copy command to copy all source fields including attachments and core fields into the target record, updating or creating that record,

ARS Table Backup and Restore

There are two scripts in this sample, one to back up a table and the other to restore a table.



```
SthMupd.exe    SvTbl.ini Do -p WebScr   bkp20111231/WebScr/a
SthMupd.exe    LdTbl.ini Do -p WebScr   bkp20111231/WebScr/a
```

To back up any ARS table, run the SvTbl.ini script passing as arguments, the table name, and a backup file prefix. The restore script will take as input the same table name and same file prefix.

The backup script will generate these files:

- a single csv containing all data from each field of the passed table
- if and only if there are attachment fields in that table, a CSV of the field names and field ids for these attachment fields
- a file prefixed by the passed prefix for each attachment.

The restore script will process these files as a set:

- a single csv containing all data from each field of the passed table
- if the attachment fields CSV exists, will read these attachment fields and ids into a script array
- if there are attachment fields, and the data CSV indicates a non-null attachment, a file saved by the backup script will update the attachment content and have the original attachment name.

This script introduces more complex features of Meta-Update. The script demonstrates:

- Query=, Output=
- Field Loops
- Output files based on schemas
- Schemas and Queries passed as arguments
- extracting and loading attachments

Running the script.

The package is in the distribution and may also be downloaded from the script library.

The package contains a def file for the form _Test. It also contains data saved by the sample save script that can be used to populate the _Test table.

To validate these scripts, simply run the backup against a single record, generate a report of all data from this record, delete the record, run the restore, generate a second report from this record, and, do a difference of the two reports. There should be no differences between the two reports.

```
SthMupd   SvTbl.ini Do -p _Test test "'1' = \"000000000000001\""
SthMqry   -f -S _Test “’1’ = \”000000000000001\”” > rpt-before.txt
SthMdel   _Test  "'1' = \"000000000000001\""
SthMupd   LdTbl.ini Do -p _Test test
SthMqry   -f -S _Test “’1’ = \”000000000000001\”” > rpt-after.txt
diff      rpt-before.txt   rpt-after.txt
```

Backup Script Overview

`[Do]` is the main command section and issues the query against the passed table. Each record is assigned to the tag `Src`.

An AssignInit is used to initialize script variables and formulate a default query string (1=1) if the script was not passed a query qualification.

`[Do]` will output a record to a CSV for each record it processes. It will not change any values other than encoding any embedded quotes and line feeds. The assignments to the output CSV are handled by a single copy command. The file’s fields are also copied from the passed table name.

`[Do] will Launch [Sv-Att-Struct]` once only.

> `[Sv-Att-Struct]` creates a second CSV containing a list of all the Attachment fields Field Names and IDs).
>
> If there are no attachment fields, the CSV is not created. The single Launch is controlled by the script variable `$V, First$` which is initialized to TRUE and set to FALSE by an AssignInit in `[Sv-Att-Struct]`.
>
> If there are any attachment fields, the CSV is created and a variable is set to indicate that there are attachments that should be saved.

`[Do]` will Launch `[Sv-Att]` each record it processes if there are any attachment fields in the table. This is controlled by the `$V, gotAtt$` script variable which was set by `[Sv-Att-Struct]`

> `[Sv-Att]` iterates through all non-null attachment fields in the `Src` record. So, for any single record it may iterate zero or more times.
>
> `[Sv-Att]` has no record or file output, so all work is done in an AssignPre section which is called after the Loop’s Tag is assigned on each iteration.
>
> The assignment is a simple AttachmentSave command issued to save the attachment to the file system. The file is named as follows:
>
> ```
> -prefix-  ReqId – FieldId .att
> ```
>
> Prefix is passed on the command line, ReqId is the request id field with any ‘|’ characters (from Join forms) translated to ‘-‘. This is done through a simple regular expression used for the side effect of allowing a Subst field specification.

Meta-Update script

```
# Meta-Update sample script file.
# Meta-Update is copyright 1996-2011 by Software Tool House Inc.
#
# File:                        SvTbl.ini
#                              Part of the sample scripts for Meta-Update.
#
#         Two scripts used to save and restore any ARS tables' data.
#         This is the Save script.  See LdTbl.ini for the restore script
#
#         This Save script will save all records into a single CSV
#         and attachments into files prefixed by the passed argument.
#---------------------------------------------------------------------



[Main]
#  The main section gives sign-on info and declares
#    Script arguments required and usage info.

Server   = $ ENV, ArsSvr  $
Port     = $ ENV, ArsPort $
User     = $ ENV, ArsUsr  $
Password = $ ENV, ArsPwd  $

PrmReq       = 2,. Function
PrmReq       =  .  Two scripts used to save and restore ARS tables.
PrmReq       =  .  This is the Save script.
PrmReq       =  .
PrmReq       =  . Usage:
PrmReq       =  .    SvTbl.ini  Do -p   tbl  outp  [ qry ]

ArgNm        = schema
ArgNm        = F-out
ArgNm        = qry

#---------------------------------------------------------------.do
[Do]
#
# This is the main entry point and called routine. This section
#   reads through the given table creating the output CSV file
#
# A Query is executed on the source table and the output file record
#   record is created using an assignment copy command.
#
# Once only, a section that saves a CSV of attachment files is
#   launched.  If there are attachment fields, a section is
#   launched each record to save those attachments to the file system.
#
```

Server connectivity and authentication set from environment variables.

```
#[Do]
#
AssignInit  = asg-I
Query       = Src,                                                   &
              $Arg, schema$,                                         &
              $V,   Qual$
Output      = Tgt,                                                   &
              Out-f,                                                 &
              $Arg,  F-out$.csv
Assign      = asg
Launch      = @if("$V, First$")  Sv-Att-Struct
Launch      = @if("$V, gotAtt$") Sv-Att


[Out-f]
#
# This declares the output CSV file.
#
Type       = Delimited, ",", FldHdr
Format     = Quoted always Quotes escape lf escape
Fields     = Out-f-flds

[Out-f-flds]
@Cmd       = Copy, $Arg, schema$

[asg-I]
#
#  This "initial" assignment section initialises script variables
#  Input Tags
#    Arg  Ptn           "" or a query string
#  Output Tags
#    V    First         do Attachment File output one time
#    V    gotAtt        table has attachments; set Sv-Att-Struct
#    V    Qual          "1=1" or the passed query string
#    V    AttPth        the attachment path
#
@Cmd        = Ref,   V,  gotAtt,     0
@Cmd        = Ref,   V,  First,      1
@Cmd        = Ref,   V,  Qual,       "1=1"
@Cmd        = Ref,   V,  AttPth,     "$Arg, F-out$"
@Cmd        = @if("$Arg, qry$" != "")                                &
              Ref,   V,  Qual,   "$Arg, qry$"
[asg]
#
# This is the assignment to the CSV file.  Because all fields
#   from the table and CSV file match, we just issue a copy
#
@Cmd       = Copy,  Src
```

The ARS Schema is a reference. As is the Query qualification.

The output file name is the passed prefix appended with ".csv"

The ARS Schema's fields are copied into the output file's definition.

This single command assigns all fields from the table to the CSV converting embedded line feeds and quotes as specified.

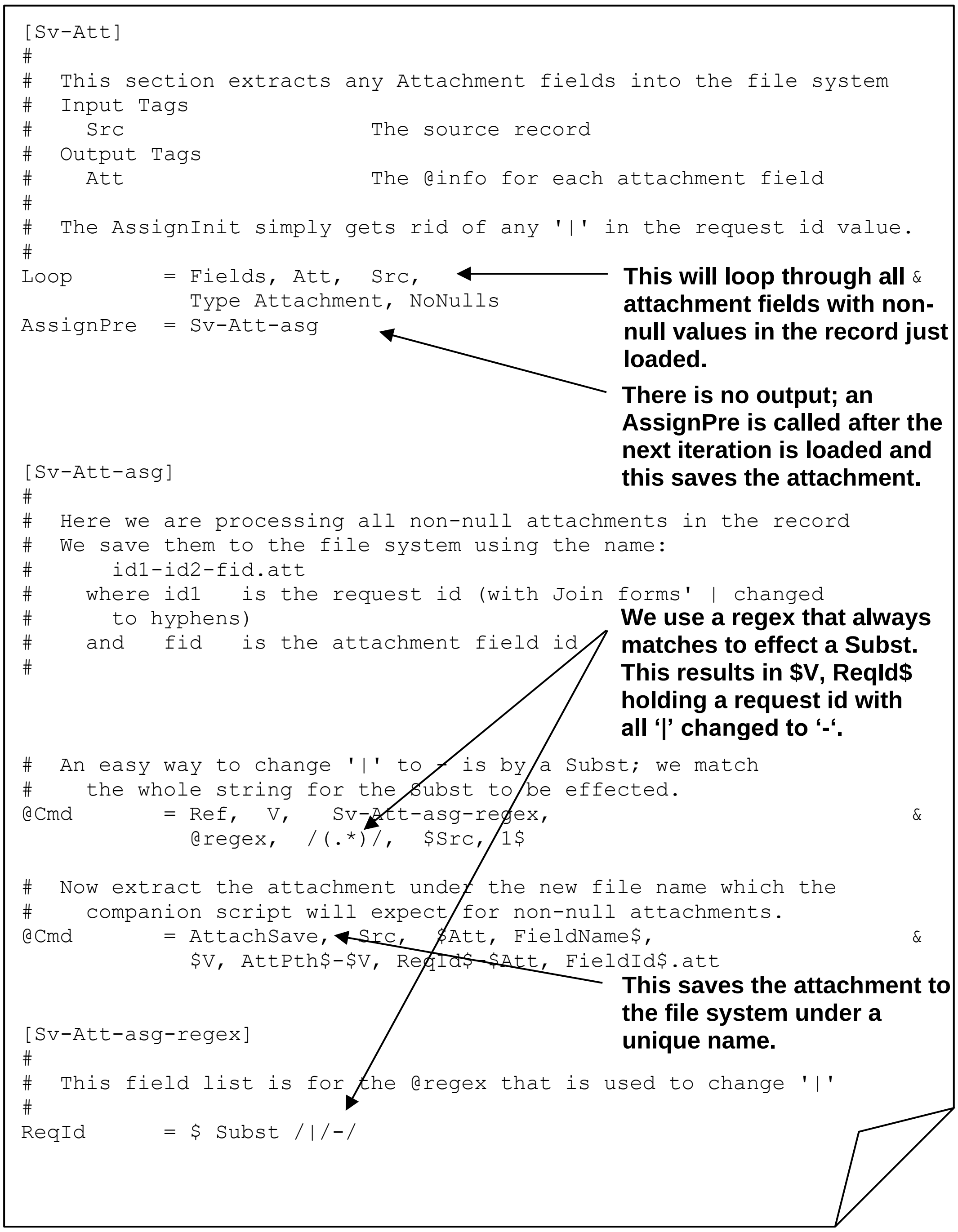
```
[Sv-Att-Struct]
#
#  This section saves the field names and ids of any attachment fields
#    into a special CSV processed by the companion script.
#
#  Input Tags
#    Src                       The source record
#  Output Tags
#    V    First              0    we want to execute once only
#    V    gotAtt             1    says we have attachment fields
#
Loop          = Fields, Att,  Src, Type Attachment
Output        = TgtS,                                                  &
                Out-f-struct,                                          &
                $Arg,  F-out$.att.csv
Assign        = Sv-Att-Struct-asg
AssignInit    = Sv-Att-Struct-asg-Init
```

If there are no attachment fields, the loop is executed zero times, no file is created, and `gotAtt` is not set true.

```
[AssignInit  = Sv-Att-Struct-asg-Init]
@Cmd         = Ref,   V,   First,     0
```

No matter if there are any attachment fields or not, we want to set `First` false.

```
[Sv-Att-Struct-asg]
@Cmd         = Ref,   V,   gotAtt,    1
AttFldNm     = Att,  FieldName
AttFldId     = Att,  FieldId



[Out-f-struct]
#
# This declares the output CSV file listing the attachment fields
#
Type         = Delimited, ",", FldHdr
Format       = Quoted always Quotes escape lf escape
Fields       = Out-f-struct-flds

[Out-f-struct-flds]
AttFldNm     = $
AttFldId     = $
```



```
[Sv-Att]
#
#  This section extracts any Attachment fields into the file system
#  Input Tags
#    Src                    The source record
#  Output Tags
#    Att                    The @info for each attachment field
#
#  The AssignInit simply gets rid of any '|' in the request id value.
#
Loop        = Fields, Att,  Src,
              Type Attachment, NoNulls
AssignPre   = Sv-Att-asg




[Sv-Att-asg]
#
#  Here we are processing all non-null attachments in the record
#  We save them to the file system using the name:
#      id1-id2-fid.att
#    where id1   is the request id (with Join forms' | changed
#      to hyphens)
#    and   fid   is the attachment field id
#



#  An easy way to change '|' to - is by a Subst; we match
#    the whole string for the Subst to be effected.
@Cmd        = Ref,  V,   Sv-Att-asg-regex,                           &
              @regex,  /(.*)/,  $Src, 1$

#  Now extract the attachment under the new file name which the
#    companion script will expect for non-null attachments.
@Cmd        = AttachSave, Src,  $Att, FieldName$,                    &
              $V, AttPth$-$V, ReqId$-$Att, FieldId$.att


[Sv-Att-asg-regex]
#
#  This field list is for the @regex that is used to change '|'
#
ReqId       = $ Subst /|/-/
```

This will loop through all & attachment fields with non-null values in the record just loaded.

There is no output; an AssignPre is called after the next iteration is loaded and this saves the attachment.

We use a regex that always matches to effect a Subst. This results in $V, ReqId$ holding a request id with all '|' changed to '-'.

This saves the attachment to the file system under a unique name.

Restore Script Overview

`[Do]` is the main command section and does no iteration or output instead only Launching two sections once.

An AssignInit is used to initialize script variables. There is no Query argument in the restore script. The AssignInit also determines if an Attachment Fields CSV exists or not. It does this with a Reference spawn assignment that assigns "OK" to the stdout variable if the file exists.

Note that because of the UNIX if shell syntax the stdout and stderr redirection does not come at the end of the command line and is explicitly stated.

`[Do] will Launch [Do-Att-Flds]` once only.

> `[Do-Att-Flds]` processes the Attachment Fields CSV just building a "script array" of Attachment Field Names and Field IDs and setting the number of attachment fields.
>
> If there are no attachment fields, the CSV was not created and the number of attachment fields remains 0.
>
> `[Do-Att-Flds]` makes no output, so only an AssignPre is used. That AssignPre section increments the number of attachment fields counter and sets the Field Name and Id into the array.

```
[Do-Att-Flds-asg]
#
#  For each field, increase the number of fields,
#    and set it in the Va, Fnm and Fid arrays
#
@Cmd        = Ref,    Va, Max, @eval, $Va, Max$+1
@Cmd        = Ref,    Va, @,    Do-asg-FF

[Do-asg-FF]
Fnm$Va, Max$   = F, AttFldNm
Fid$Va, Max$   = F, AttFldId
```

Increment `Va, Max`

This assigns a series of "field / value" pairs to the `Va` tag. We use references in the fields to be assigned.

Tags built are like this:

```
Va,    Max       2
Va,    Fnm1          Attachment1
Va,    Fid1          5378001021
Va,    Fnm2          Attachment2
Va,    Fid2          5378001022
```

`[Do]` then Launches `[Do-Load]`, the backup file handling section, since all Attachment fields are now known.

> `[Do-Load]` Processes the passed backup data file and updates the passed table using '1' = the first field of the file" as the update query.
>
> Like the backup script, the File's fields are copied from the schema and the schema in the query and the file's field's copy is the `$Arg, schema$` reference.
>
> Because the file's fields are copied, the file's field 1 is the first schema field, or field '1', and this is used in the Update= query.

The Update is done with the Merge API and with Merge workflow inhibited. It is only through the Merge API that core fields may be set (such as Request ID, Submitter, Create Date).

Note that this restore script will not work with Join forms unless Merge workflow is allowed. A write to a join can only write to the database if the filters on that join fire.

The Assignment section for the ARS Table Update= is the same for new or updated records.

If there are any attachment fields and the backup data indicates that it is non-null, a string is assigned with two file names:

```
original attachment name, attachment file

C:\dir\xxx.xxx, -prefix-  ReqId – FieldId .att
```

Meta-Update can process attachment values as references, single file strings, or double file strings. In the case of a double file string, the second string is the file in the file system that contains the data of the attachment, and the first name is the file name set into the attachment value.

Because the file is copied from the table, a simple copy assignment command will set all fields to their backed up values skipping any fields that have already been assigned a value.

Meta-Update script

```
# Meta-Update sample script file.
# Meta-Update is copyright 1996-2011 by Software Tool House Inc.
#
# File:                        LdTbl.ini
#                              Part of the sample scripts for Meta-Update.
#
#         Two scripts used to save and restore any ARS tables' data.
#         This is the Load script.  See SvTbl.ini for the backup script.
#
#         This Load script will process the CSV files generated by the
#         save script and load all records including any attachments
#---------------------------------------------------------------------

[Main]
#  [Main] gives sign-on info and declares Script arguments.
Server    = $ ENV, ArsSvr  $
PrmReq       =  .   LdTbl.ini  Do -p   tbl  outp

ArgNm        = schema
ArgNm        = F-inp
#---------------------------------------------------------------.do
[Do]
#
# Before we can proceed with loading the data file, we'll need a list
#   of Attachment fields so that we can assign them as needed.
#
# So, here, the AssignInit figures out if the attachment fields CSV
#   exists, then, launches [Do-Att-Flds] to save attachment fields in
#   script variables, and finally launch the Do-Load section to process
#   the backup data file against the ARS table.
#
AssignInit   = asg-I
Launch       = @if("$Va, Do$") Do-Att-Flds
Launch       = Do-Load


[asg-I]
#
#  This "initial" assignment section sets $Va, Do$ to the existence of
#    the "$Arg, F-inp$.att.csv" file and makes a few initializations.

#  Input Tags
#    Arg     F-inp       the output file name
#  Output Tags
#    Va      Max        init num attachment fields to 0
#    Va      Do         set to true if file $Arg, F-inp$.att.csv exists.
#
@Cmd        = Ref,   Va, Max,       0
@Cmd        = Ref,   Va, Do,        0
@Cmd        = @if("$CTL, OS$" == "UNIX")
  @Cmd        = Ref,   V,  @spawn,                                       &
        if [ -f '$Arg, F-inp$.att.csv' ] ;                               &
        then echo OK $redir$ ;                                           &
        fi;
@Cmd        = else
  @Cmd        = Ref,   V,  @spawn,                                       &
        if exist "$Arg, F-inp$.att.csv" echo OK
@Cmd        = endif
@Cmd        = @if("$V, stdout$" ~= "OK")
  @Cmd        = Ref,   Va, Do,        1
@Cmd        = endif
```

The AssignInit section `[asg-I]` sets `$Va, Do$` to true if the Attachment Fields CSV exists in the expected location.

Note different command to determine file existence n Windows and Unix. Note use of `$redir$` in Unix command.

The echo produces "`OK<lf>`" or "`OK<cr><lf>`" in `$V, stdout$`, so we just check for a leading OK.

```
 [Do-Att-Flds]
#
#  The SvTbl companion script generated an attachment fields CSV.
#    We are only Launched if this file exists!
#  We set number of attachment fields for the Update= assignments.
#
#  Input Tags
#    Va   Max              0        number of attachment fields
#  Output Tags
#    Va   Max              0 + n    number of attachment fields
#    Va   Fnm1,2, ..       char     field name array 1..n
#    Va   Fid1,2, ..       int      field id   array 1..n
#
File         = F,                                                  &
               Inp-f-att,                                          &
               $Arg,  F-inp$.att.csv
AssignPre    = Do-Att-Flds-asg

[Do-Att-Flds-asg]
#
#  For each field, increase the number of fields, and set it in the
#    Va, Fnm and Fid arrays
#
@Cmd        = Ref,   Va, Max,       @eval,   $Va, Max$+1
@Cmd        = Ref,   Va, @,         Do-asg-FF

[Do-asg-FF]
Fnm$Va, Max$   = F, AttFldNm
Fid$Va, Max$   = F, AttFldId


# File declarations:  the two input CSV files
#   Inp-f-att        saved by SvTbl.ini; schema's attachment fields

[Inp-f-att]
#
Type        = Delimited, ",", FldHdr
Format      = Quoted always Quotes escape lf escape
Fields      = Inp-f-att-flds

[Inp-f-att-flds]
AttFldNm    = $
AttFldId    = $
```

Increment `Va, Max`

This assigns a series of "field / value" pairs to the `Va` tag. We use references in the fields to be assigned to build an array.

```
[Do-Load]
#
# Loops through the given CSV (created by the companion script)
#   updating in the target table with the value of the first CSV
#   field (Request ID) being matched against '1'
#
# We need to use Merge (like the Import Tool) so that we can assign
#   core fields like '1' etc. For Joins, remove NoWorkflow from Merge.
#
#  We know the number of attachment fields, their names, and ids, so
#    if the attachment fields are non-null, they are assigned with
#    their original file name and the expected file system name.
#
#  The remaining field values are simply copied from the CSV row.
#
File         = Src,
               Inp-f,
               $Arg,  F-inp$.csv
Update       = Tgt,
               $Arg, schema$,
               '1' = "$Src, 1$"
AssignNew    = Do-Load-asg
Assign       = Do-Load-asg
Merge        = Yes, NoWorkflow


[Do-Load-asg]
#
# This is the assignment to the ARS record from the CSV file
#   (with the same fields as the ARS record).  Because all fields
#   from the table and CSV file match, we just issue a copy.
#
# We need the CoreAssign option because we want '1', '2', etc assigned
from the CSV - only available with Merge
#
#  If the attachment value in the CSV is non-null, we will have a
#    file named:   id1-id2-fid.att
#     id1 etc    is the request id (with '|' changed to hyphens)
#     fid        is the attachment field id

#  We change '|' to - with a Subst; we match all for the Subst
@Cmd        = Ref,  V,   Ld-Att-asg-regex,  @regex,  /(.*)/,  $Src, 1$



[Ld-Att-asg-regex]
#
#  This field list is for @regex used to substitute hyphens for '|'
#
ReqId       = $ Subst /|/-/
```

We use the reference `$Src, 1$` to indicate the first CSV field which will be Request ID, Entry ID, and so on.

You cannot use `NoWorkflow` on Join forms.

```
[Do-Load-asg]

# handle attachments separately
@Cmd         = @if("$Va, Max$")
  @Cmd         = Ref,   V,   @info,    Src,   $Va, Fnm1$
  @Cmd         = @if("$V, Value$")
    @Cmd         = Ref,   V,   attval,                                   &
           "$V,   Value$,$V, AttPth$-$V, ReqId$-$Va, Fid1$.att"
    $V, FieldName$ =   V, attval
  @Cmd         = endif
  @Cmd         = @if("$Va, Max$" > 1)
    @Cmd         = Ref,   V,   @info,    Src,   $Va, Fnm2$
    @Cmd         = @if("$V, Value$")
      @Cmd         = Ref,   V,   attval,                                 &
                 "$V,   Value$,$V, AttPth$-$V, ReqId$-$Va, Fid2$.att"
      $V, FieldName$ =   V, attval
    @Cmd         = endif
    @Cmd         = @if("$Va, Max$" > 2)
      @Cmd         = Ref,   V,   @info,    Src,   $Va, Fnm3$
      @Cmd         = @if("$V, Value$")
        @Cmd         = Ref,   V,   attval,                               &
                   "$V,   Value$,$V, AttPth$-$V, ReqId$-$Va, Fid3$.att"
        $V, FieldName$ =   V, attval
      @Cmd         = endif
      @Cmd         = @if("$Va, Max$" > 3)



      @Cmd         = endif
    @Cmd         = endif
  @Cmd         = endif
@Cmd         = endif


@Cmd         = Copy,   Src,    CoreAssign
```

The maximum number of attachment fields in any one form should be handled here, with, perhaps, an error thrown if it is exceeded.

The remaining assignments are handled with a Copy command.

Index

Index