Y-Innovate Build System for z/OS

Bobby Tjassens Keiser

12 April 2017

Agenda

- Introduction
- What is LWZMAKE?
  - Introduction
  - Why show it at REXXLA Symposium?
  - Detailed example explained
- Demo's
  - Deployment automation
  - Build automation
- Q&A

Introduction

- Bobby Tjassens Keiser
  38 yrs, married, 3 kids
  Co-founder of Y-Innovate IT, an ISV and IBM business partner
  Employee with ICU IT Services
  Z-enthusiast
- Y-Innovate IT
  Creator of Light Weight Web framework for CICS (LWW), a product to support web development with CICS on z/OS.
  Recent side project: LWZMAKE

Y-Innovate
Innovations for the Enterprise

ICU
CHALLENGING IT

light weight web
Framework for CICS®

What is LWZMAKE?
Introduction

- New build automation tool, loosely based on **`make`** (well known in the *nix world)
- Specific for Z System platform, emphasis on traditional 'MVS' environment (PDS(E)'s, members, sequential data sets, that sort of thing)
- **Open source!** Get it at: https://github.com/Y-Innovate/LWZMAKE
- Combination of a single Assembler source, which results in a single load module, and a set of sample JCL's to run it and sample REXX EXECs to perform build functions.

What is LWZMAKE?
More introduction

- Just like **`make`** does, **`LWZMAKE`** can 'update files from others whenever the others change'.
  e.g. only copy members from source PDS's to target PDS's when the source PDS's were altered more recently.
- Also just like with **`make`**, the way to tell the utility what to do is with a script in **`LWZMAKE`**'s script language. Such a script is often called a **`makefile`** (again loosely based on **`make`**'s script syntax).

What is LWZMAKE?
Why show it at the REXXLA Symposium?

- Unlike **`make`**, instead of firing off command lines for performing build activities, you call REXX EXECs to do those things.
- For example in the following makefile, the CALL statement at the bottom invokes a REXX EXEC called IEBCOPY, which in turn invokes the IEBCOPY utility.

```
01| .RECIPEPREFIX = -
02|
03| srchlq  := SOMEUSR
04| tgthlq  := MYUSR
05| targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07| .PHONY ALL
08| ALL : $(targets)
09|
10| # Copy MEM1 and MEM2, but only if they changed
11| $(targets) : $(srchlq).PDS.JCL($%)
12| - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13| -              MEMBER($%)
```

What is LWZMAKE?
Why show it at the REXXLA Symposium? (continued)

- The reason to show it at the REXXLA Symposium is because of the tight relation to REXX (so I'm hoping you'll find it interesting). **LWZMAKE** determines which files require a build and invokes one or more REXX EXECs to perform the actual build tasks. Those REXX EXECs can focus on a single file instead of listing PDS members etc.
- Also I'm hoping to get feedback (what I really want is for you to download it, use it, tell me what could be improved or added, contribute your own REXX's etc).

What is LWZMAKE?
Explaining the example

- Going back to the example:

```
01| .RECIPEPREFIX = -
02|
03| srchlq  := SOMEUSR
04| tgthlq  := MYUSR
05| targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07| .PHONY ALL
08| ALL : $(targets)
09|
10| # Copy MEM1 and MEM2, but only if they changed
11| $(targets) : $(srchlq).PDS.JCL($%)
12| - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL)  DSOUT($(tgthlq).PDS.JCL) \
13| -              MEMBER($%)
```

These are variable assignments

This is a 'rule'

This is another 'rule'

This is a 'recipe'

Comments start with #

Line continuation with \

What is LWZMAKE?
Explaining the example

Assignment of special variable to tell that recipes start with -

- Going back to the example:

Direct assignment := means variables are resolved immediately

Variables used by enclosing in $(...)

```
01| .RECIPEPREFIX = -
02|
03| srchlq  := SOMEUSR
04| tgthlq  := MYUSR
05| targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07| .PHONY ALL
08| ALL : $(targets)
09|
10| # Copy MEM1 and MEM2, but only if they changed
11| $(targets) : $(srchlq).PDS.JCL($%)
12| - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13| -              MEMBER($%)
```

What is LWZMAKE?
Explaining the example

- Going back to the example:

```
01| .RECIPEPREFIX = -
02|
03| srchlq  := SOMEUSR
04| tgthlq  := MYUSR
05| targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07| .PHONY ALL
08| ALL : $(targets)
09|
10| # Copy MEM1 and MEM2, but only if they changed
11| $(targets) : $(srchlq).PDS.JCL($%)
12| - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13| -              MEMBER($%)
```

PHONY targets don't represent real files, but are used to get prerequisites built

Rules consist of one or more targets left of the :

...and one ore more prerequisites right of the :

Special variables $@ and $% mean “current target” and “current target's PDS member”

What is LWZMAKE?
Explaining the example

- Going back to the example:

```
01|  .RECIPEPREFIX = -
02|
03|  srchlq  := SOMEUSR
04|  tgthlq  := MYUSR
05|  targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07|  .PHONY ALL
08|  ALL : $(targets)
09|
10|  # Copy MEM1 and MEM2, but only if they changed
11|  $(targets) : $(srchlq).PDS.JCL($%)
12|  - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13|  -               MEMBER($%)
```

Recipes can contain variable assignment and CALL statements

CALL invokes a REXX EXEC named by the 1st parameter

Anything beyond that 1st parameter is passed to the REXX EXEC as an argument

What is LWZMAKE?
2 phases of execution

- LWZMAKE processes a makefile in 2 phases
- During the first phase
  - the **`makefile`** is parsed and committed to memory.
  - Variables are assigned their values.
  - Variables are resolved when
    - referred to in direct assignments :=
    - or in targets (left of the : in rules)
  - Variables referred to in prerequisites or in recipes are left unresolved.

```
01|  .RECIPEPREFIX = -
02|
03|  srchlq  := SOMEUSR
04|  tgthlq  := MYUSR
05|  targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07|  .PHONY ALL
08|  ALL : $(targets)
09|
10|  # Copy MEM1 and MEM2, but only if they changed
11|  $(targets) : $(srchlq).PDS.JCL($%)
12|  - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PD
13|  -              MEMBER($%)
```

What is LWZMAKE?
2 phases of execution

- So for our example, after the first phase:
  - These variables are in memory:

variable	*value*
srchlq	SOMEUSR
tgthlq	MYUSR
targets	MYUSR.PDS.JCL(MEM1) MYUSR.PDS.JCL(MEM2)

```
01| .RECIPEPREFIX = -
02|
03| srchlq  := SOMEUSR
04| tgthlq  := MYUSR
05| targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07| .PHONY ALL
08| ALL : $(targets)
09|
10| # Copy MEM1 and MEM2, but only if they changed
11| $(targets) : $(srchlq).PDS.JCL($%)
12| - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13| -              MEMBER($%)
```

  - These targets are in memory:

target	*prerequisites*	*recipe*
ALL	$(targets)	
MYUSR.PDS.JCL(MEM1)	$(srchlq).PDS.JCL($%)	- CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \ - MEMBER($%)
MYUSR.PDS.JCL(MEM2)	$(srchlq).PDS.JCL($%)	- CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \ - MEMBER($%)

What is LWZMAKE?
2 phases of execution

- During the second phase
  - the requested (or first found) target is processed by
    - resolving any variables in its prerequisites
    - looking up every prerequisite to see if there are targets defined for them
    - if so, recursively process those targets first
    - when any of the prerequisites are altered at a later date+time than the target, that target requires a build
    - so then the variables in the accompanying recipe are resolved
    - and the recipe is executed

```
01|  .RECIPEPREFIX = -
02|
03|  srchlq  := SOMEUSR
04|  tgthlq  := MYUSR
05|  targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07|  .PHONY ALL
08|  ALL : $(targets)
09|
10|  # Copy MEM1 and MEM2, but only if they changed
11|  $(targets) : $(srchlq).PDS.JCL($%)
12|  - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PD
13|  -               MEMBER($%)
```

What is LWZMAKE?
2 phases of execution

- So for our example, the second phase results in:
  - The first target ALL is processed:

Target	ALL
Prerequisites	$(targets)
Recipe	

```
01|  .RECIPEPREFIX = -
02|
03|  srchlq  := SOMEUSR
04|  tgthlq  := MYUSR
05|  targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07|  .PHONY ALL
08|  ALL : $(targets)
09|
10|  # Copy MEM1 and MEM2, but only if they changed
11|  $(targets) : $(srchlq).PDS.JCL($%)
12|  - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13|  -               MEMBER($%)
```

What is LWZMAKE?
2 phases of execution

- So for our example, the second phase results in:
  - Variables in its prerequisites are resolved:

Target	ALL
Prerequisites	MYUSR.PDS.JCL(MEM1) MYUSR.PDS.JCL(MEM2)
Recipe	

```
01|  .RECIPEPREFIX = -
02|
03|  srchlq  := SOMEUSR
04|  tgthlq  := MYUSR
05|  targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07|  .PHONY ALL
08|  ALL : $(targets)
09|
10|  # Copy MEM1 and MEM2, but only if they changed
11|  $(targets) : $(srchlq).PDS.JCL($%)
12|  - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13|  -               MEMBER($%)
```

What is LWZMAKE?
2 phases of execution

- So for our example, the second phase results in:
  - The first prerequisite MYUSR.PDS.JCL(MEM1) is looked up and found as a target:

Target	MYUSR.PDS.JCL(MEM1)
Prerequisites	$(srchlq).PDS.JCL($%)
Recipe	- CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \ - MEMBER($%)

```
01|  .RECIPEPREFIX = -
02|
03|  srchlq  := SOMEUSR
04|  tgthlq  := MYUSR
05|  targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07|  .PHONY ALL
08|  ALL : $(targets)
09|
10|  # Copy MEM1 and MEM2, but only if they changed
11|  $(targets) : $(srchlq).PDS.JCL($%)
12|  - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13|  -              MEMBER($%)
```

What is LWZMAKE?
2 phases of execution

- So for our example, the second phase results in:
  - Variables in its prerequisites are resolved:

Target	MYUSR.PDS.JCL(MEM1)
Prerequisites	SOMEUSR.PDS.JCL(MEM1)
Recipe	- CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \ - MEMBER($%)

```
01|  .RECIPEPREFIX = -
02|
03|  srchlq  := SOMEUSR
04|  tgthlq  := MYUSR
05|  targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07|  .PHONY ALL
08|  ALL : $(targets)
09|
10|  # Copy MEM1 and MEM2, but only if they changed
11|  $(targets) : $(srchlq).PDS.JCL($%)
12|  - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13|  -              MEMBER($%)
```

What is LWZMAKE?
2 phases of execution

- So for our example, the second phase results in:
  - The prereq is looked up, not found as a target, assumed an existing file. If prereq updated more recently than target, then the variables in the recipe are resolved:

Target	MYUSR.PDS.JCL(MEM1)
Prerequisites	SOMEUSR.PDS.JCL(MEM1)
Recipe	- CALL IEBCOPY PDSIN(SOMEUSR.PDS.JCL) PDSOUT(MYUSR.PDS.JCL) \ - MEMBER(MEM1)

```
01|  .RECIPEPREFIX = -
02|
03|  srchlq  := SOMEUSR
04|  tgthlq  := MYUSR
05|  targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07|  .PHONY ALL
08|  ALL : $(targets)
09|
10|  # Copy MEM1 and MEM2, but only if they changed
11|  $(targets) : $(srchlq).PDS.JCL($%)
12|  - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13|  -              MEMBER($%)
```

What is LWZMAKE?
2 phases of execution

- So for our example, the second phase results in:
  - Then the recipe is executed, in this case invoking IEBCOPY to copy member MEM1 from SOMEUSR.PDS.JCL to MYUSR.PDS.JCL:

```
****************************** Top of Data ******************************
/* REXX */
/*********************************************************************/
/* Program     : IEBCOPY                                             */
/*                                                                   */
/* Description: This program invokes IEBCOPY to copy one, multiple or */
/*              all members from one PDS(E) to another.              */
```

```
01| .RECIPEPREFIX = -
02|
03| srchlq  := SOMEUSR
04| tgthlq  := MYUSR
05| targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07| .PHONY ALL
08| ALL : $(targets)
09|
10| # Copy MEM1 and MEM2, but only if they changed
11| $(targets) : $(srchlq).PDS.JCL($%)
12| - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13| -              MEMBER($%)
```

What is LWZMAKE?
2 phases of execution

- So for our example, the second phase results in:
  - The processing of ALL's first prerequisite is finished, now follows the same processing for the second, possibly resulting in MEM2 begin copied:

Target	ALL
Prerequisites	MYUSR.PDS.JCL(MEM1) MYUSR.PDS.JCL(MEM2)
Recipe	

```
01|  .RECIPEPREFIX = -
02|
03|  srchlq  := SOMEUSR
04|  tgthlq  := MYUSR
05|  targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
06|
07|  .PHONY ALL
08|  ALL : $(targets)
09|
10|  # Copy MEM1 and MEM2, but only if they changed
11|  $(targets) : $(srchlq).PDS.JCL($%)
12|  - CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
13|  -               MEMBER($%)
```

Agenda

- Introduction
- What is LWZMAKE?
  - Introduction
  - Why show it at REXXLA Symposium?
  - Detailed example explained
- **Demo's**
  - **Deployment automation**
  - **Build automation**
- Q&A

Q & A

?