Y-Innovate Build System for z/OS

Bobby Tjassens Keiser
12 April 2017

Y-Innovate
Innovations for the Enterprise
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
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.

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

- These are variable assignments
- Comments start with #
- Line continuation with \

This is a 'rule'
This is another 'rule'
This is a 'recipe'
What is LWZMAKE?
Explaining the example

Going back to the example:

```
.RECIPEPREFIX =

srchq := SOMEUSR

tgthq := MYUSR

targets := $(tgthq).PDS.JCL(MEM1) $(tgthq).PDS.JCL(MEM2)

.PHONY ALL

ALL : $(targets)

# Copy MEM1 and MEM2, but only if they changed
$(targets) : $(srchq).PDS.JCL($%)

- CALL IEBCOPY PDSIN($(srchq).PDS.JCL) PDSOUT($(tgthq).PDS.JCL) MEMBER($%)
```
What is LWZMAKE?
Explaining the example

Going back to the example:

Rules consist of one or more targets left of the :

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

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

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| srcql := SOMEUSR
04| tgtql := MYUSR
05| targets := $(tgtql).PDS.JCL(MEM1) $(tgtql).PDS.JCL(MEM2)
06| 07| .PHONY ALL
08| 09| ALL : $(targets)
10| # Copy MEM1 and MEM2, but only if they changed
11| $(targets) : $(srcql).PDS.JCL($%)
12|  - CALL IEBCOPY PSIN($(srcql).PDS.JCL) PDSOUT($(tgtql).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.
What is LWZMAKE?
2 phases of execution

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

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcqlq</td>
<td>SOMEUSR</td>
</tr>
<tr>
<td>tgthlq</td>
<td>MYUSR</td>
</tr>
<tr>
<td>targets</td>
<td>MYUSR.PDS.JCL(MEM1) MYUSR.PDS.JCL(MEM2)</td>
</tr>
</tbody>
</table>

  These targets are in memory:

<table>
<thead>
<tr>
<th>target</th>
<th>prerequisites</th>
<th>recipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL $(targets)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MYUSR.PDS.JCL(MEM1)</td>
<td>$(srcqlq).PDS.JCL($%)</td>
<td>CALL IEBCOPY PDSIN($(srcqlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \ MEMBER($%)</td>
</tr>
<tr>
<td>MYUSR.PDS.JCL(MEM2)</td>
<td>$(srcqlq).PDS.JCL($%)</td>
<td>CALL IEBCOPY PDSIN($(srcqlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \ MEMBER($%)</td>
</tr>
</tbody>
</table>
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
What is LWZMAKE?
2 phases of execution

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

<table>
<thead>
<tr>
<th>Target</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prerequisites</td>
<td>$(targets)</td>
</tr>
<tr>
<td>Recipe</td>
<td></td>
</tr>
</tbody>
</table>

```c
#RECIPEPREFIX = -
srclq := SOMEUSR
tgthlq := MYUSR
targets := $(tgthlq).PDS.JCL(MEM1) $(tgthlq).PDS.JCL(MEM2)
.PHONY ALL
ALL : $(targets)
# Copy MEM1 and MEM2, but only if they changed
$(targets) : $(srchlq).PDS.JCL($%
- CALL IEBCOPY PDSIN($(srchlq).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) MEMBER($%)
```
What is LWZMAKE?
2 phases of execution

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

<table>
<thead>
<tr>
<th>Target</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prerequisites</td>
<td>MYUSR.PDS.JCL(MEM1) MYUSR.PDS.JCL(MEM2)</td>
</tr>
<tr>
<td>Recipe</td>
<td></td>
</tr>
</tbody>
</table>

```
01| .RECIPEPREFIX = -
02| 03| srcql := 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) : $(srcql).PDS.JCL($%)
12| - CALL IEBCOPY PDSIN$(srcql).PDS.JCL) PDSOUT$(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   $(src1q).PDS.JCL($%)
Recipe          - CALL IEBCOPY PDSIN($(src1q).PDS.JCL) PDSOUT($(tgthlq).PDS.JCL) \
                - MEMBER($%)
```

```
01] .RECIPEPREFIX = -
02] 
03] src1q := 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) : $(src1q).PDS.JCL($%)
12] - CALL IEBCOPY PDSIN($(src1q).PDS.JCL) PDSOUT($(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:

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

```
01| .RECIPEPREFIX = -
02| 03| srchlq := SOMEUSR
04| tghlq := MYUSR
05| targets := $(tghlq).PDS.JCL(MEM1) $(tghlq).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($(srchlq).PDS.JCL) PDSOUT($(tghlq).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:
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:

<table>
<thead>
<tr>
<th>Target</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prerequisites</td>
<td>MYUSR.PDS.JCL(MEM1) MYUSR.PDS.JCL(MEM2)</td>
</tr>
<tr>
<td>Recipe</td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
01] .RECIPEPREFIX = -
02] 
03] srchqlq := SOMEUSR
04] tgthqlq := MYUSR
05] targets := $(tgthqlq).PDS.JCL(MEM1) $(tgthqlq).PDS.JCL(MEM2)
06] 
07] .PHONY ALL
08] ALL : $(targets)
09] 
10] # Copy MEM1 and MEM2, but only if they changed
11] $(targets) : $(srchqlq).PDS.JCL($%)
12] - CALL IECBCOPY PDSIN$(srchqlq).PDS.JCL) PDSOUT$(tgthqlq).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