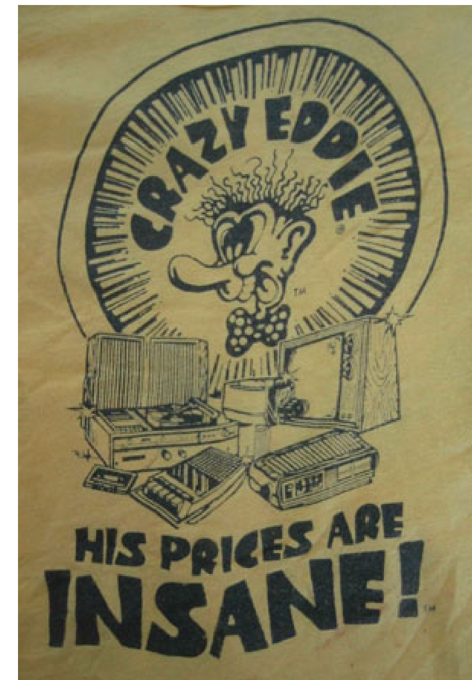
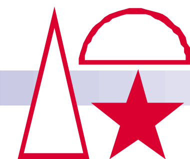
We've Gone Cra-a-a-a-zy



Alex Kodat

Sirius Software Inc.

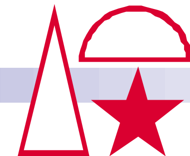


Yes, We've Gone Cra-a-a-a-zy

- We're giving away features we wrote for specific products
  - To (almost) all our customers
  - For **free**!
  - Someone get Gary a drink
- Put us in straitjackets

Why We Don't Think We're Really Crazy

- By giving away stuff we protect our customer base
  - By making it easier for people to do their jobs
  - By crushing our rivals
- The things we're giving away are very useful but almost impossible for people to cost-justify
  - So add relatively little to cost-justifiability of containing product
- We want to use the things we're giving away in our own code and in examples
  - Can do this without making it available to everyone, but a pain
- Some of the giveaways might whet your appetite for products you have to pay for
  - Oh darn



Who Gets These Features?

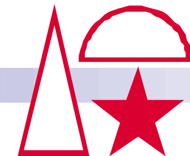
- Anyone who has Sirius Mods
  - UL API customers – SOAP, Web, Sockets, $funload, etc..
  - Fast/Reload customers
  - Fast/Backup customers
  - UL product customers – SirPro, SirMon, SirScan, etc..
  - SirFact customers
- We're not going to take it back from anyone
  - So use features in good health

Giveaway 1 – Mixed Case UL Support

- Available in Sirius Mods 6.6
- Accomplished by translating unquoted tokens to upper case
  - So “For each recOrd” converted to “FOR EACH RECORD”
- So case-insensitive mixed case support
- Mostly compile-time only
- Documented in the *Janus SOAP Reference*

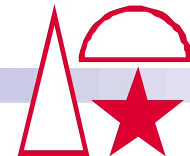
Enabling Mixed Case Support

- Mixed case Begin statement – b, begin, Begin, beGin
  - Or mixed case More
- `Sirius Case ToUpper` directive
  - Only applies to current procedure and procedures INCLUDEd by current procedure
  - But does not apply to procedure that INCLUDEd the current procedure
  - `Sirius Case Leave` turns off mixed case support
- In Sirius Mods 6.8 you can globally enable mixed case support with the COMPOPT system parameter



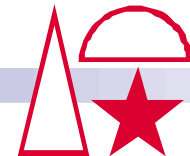
The COMPOPT System Parameter

- X'01' bit says start all requests in mixed case (ToUpper) mode
  - But `Sirius Case Leave` still honored
- X'02' bit says ignore the `Sirius Case Leave` directive
- X'04' bit says convert image and image item names in most Sirius $functions and methods to upper case at run-time
  - COMPOPT a bit of a misnomer
  - More bits probably added in the future for fieldname and screen item name variables



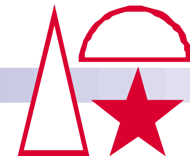
Why Use Mixed Case Support?

- Code readability
  - Use camel or Pascal notation for variable names
    - %customerIBirthDate rather than %CUSTOMERBIRTHDATE or %CUSTOMER_BIRTH_DATE or %CBDT
  - Mixed case generally easier on the eye
- Makes 204 look more modern
  - Show a 25-year old programmer all upper case code and prepare for gagging sounds
  - Put UL code next to Java or VB.net and show it to anyone (especially a manager) and all upper case UL code will just look dated
    - Even if very similar statements used
- Once you get used to mixed case UL, the upper case code will look old and clunky



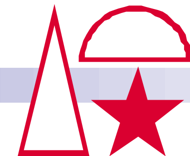
Giveaway 2 – The Assert Statement

- Syntax similar to If statement
- But causes request cancellation if condition is false
- Displays procedure and line number of Assert statement in cancellation message
  - If you own SirFact, you also get a SirFact dump
- Continue option to issue error message, but continue
- Snap option to get CCASNAP
  - Useful for Sirius problem diagnosis
- Documented in the *Janus SOAP Reference* and the *Sirius Functions Reference*



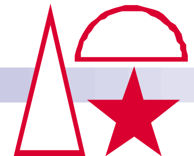
Why Would You Want to Use Assert?

- To validate assumptions inside of complicated code, or code called in many different places
  - An assumption is the mother of a foul-up
  - Better to cancel request and (possibly) back out, rather than continue with bad data
- As internal documentation of local code requirements
  - More believable than a comment
- Programmer comfort
  - “I **think** this condition will always be true here, but I'll put in an Assert to make sure”
- In general, increases code reliability



Assert Statement Example

```
Subroutine nasty(%crn is string len 9)

%found is float

f: in file customer find records for which
      crn = %crn
   end find

%found = 0

for each record in f
   %found = 1
   ...
end for

* Make sure we found the customer

assert %found

end subroutine
```



Sirius Software, Inc.

Giveaway 3 – A UL Macro Language

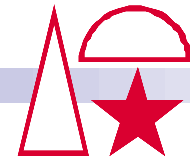
- What's a macro language?
  - It's a language that gets interpreted (executed) by the compiler
  - Most useful for tuning what gets compiled based on the environment it gets compiled in
  - Taking it further however (which we haven't (yet)) can provide powerful programming capabilities that aren't available with a pure compiled language
    - LISP is an example of a language that does this

The Sirius Macro Language Facility

- Directives (statements) that begin with exclamation mark
  - Called *bang* by the geeks - “!ifdef” said as “bang ifdef”
  - Case-sensitivity controlled the same way as other UL statements
- User language code skipped or compiled under control of macro facility
- Macro variables
  - Can be used the same way as globals

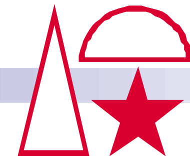
Macro Variables

- Have two *states*: defined or undefined
  - Always start out compilation as undefined
  - Defined with !def directive followed by variable name
  - Can be undefined with !undef directive
- Defined variables can have values
  - Variable value follows variable name can on !def directive
  - If not value defined, variable value is null
- To have the value of a macro variable placed in UL code (or in another macro directive) place the variable name preceded by ?! in the code
  - These are case-sensitive (so usually need to be in upper case)



Macro Variable Overhead

- Macro variables kept in CCATEMP
  - In internal $list, natch
- Discarded after compilation completes
  - Even for pre-compiled procs
  - Since unneeded after compilation
- So no run-time overhead for macro variables



Macro Variable Example

```
Begin

!def praud ?&PRAUD

?!PRAUD 'Printing or auditing'
?!PRAUD 'You decide'

print 'The value of my macro variable is ?!PRAUD'

end
```

Conditional Compilation Using Macro Statements

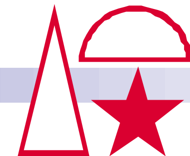
- !ifdef says compile if macro variable defined
- !ifndef says compile if macro variable not defined
- !else, !elseifdef, and !elseifndef work as one might expect
- All these directives followed by macro variable name
- No comparisons currently for macro variable value

Conditional Assembly Example

```
Begin

!def ?&DEVELOPMENT
!def ?&TEST

!ifdef ?!DEVELOPMENT
  ...  development specific stuff
!elseifdef ?!TEST
  ...  test region specific stuff
!else
  ...  production specific stuff
!endif

 ...

end
```

Why Would You Use Macro Variables and Conditional Compilation Directives?

- Easier to read than traditional “dummied” out code
- Make Include statement parameters more usable
  - Include statement parameters replace ?? with next token on Include statement
  - So no way to use a parameter multiple times
  - Also no way to use parameters out of order
- Can be used to prevent compiling the same code multiple times

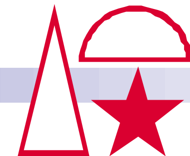


Better Include Parameter Support Example

```
PROCEDURE FOO

!def FOO.PARM1 ??
!def FOO.PARM2 ??
...
print 'The value of %?!FOO.PARM2 is ' %?!FOO.PARM2
...
if %?!FOO.PARM1 then
   ...
end if
...
END PROCEDURE
```

Including Subroutine (or Methods): The Problem

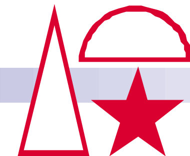
- You usually only want to Include the procedure that contains a subroutine when you're going to use it
- The same is true when you're writing a complex subroutine
- So if a complex subroutine needs to use another subroutine it needs the procedure that contains that subroutine Included
  - So it Includes it itself?
    - But what if someone else already Included the procedure?
    - Then you get compile errors
  - So outer level proc has to do Include for subroutine's procs?
    - Yuck



What One Wants to Do

```
PROCEDURE SUBROUTINE.FOO

in ?&procfile include subroutine.doit

subroutine foo
...
call doit(%whatever, 22)
...
end subroutine
END PROCEDURE

PROCEDURE SUBROUTINE.BAR

in ?&procfile include subroutine.doit

subroutine bar
...
call doit(%whatever, 22)
...
end subroutine
END PROCEDURE
```

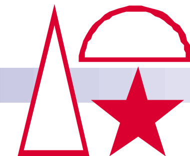
But what if you want to use both Foo and Bar in the same request?



A Solution Using Macro Variables The Way It's Done in C

```
PROCEDURE SUBROUTINE.DOIT
!ifndef subroutine.doit
!def subroutine.doit

subroutine doit
...
end subroutine
!endif
END PROCEDURE
```

Not really required:
End of procedure
closes out all !if blocks



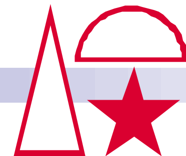
Sirius Software, Inc.

But This is Such a Common Usage !Dupexit Shortcut

```
PROCEDURE SUBROUTINE.DOIT
!dupexit

subroutine doit
...
end subroutine
END PROCEDURE
```

!Dupexit

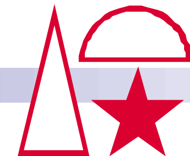
- Does two things:
  - Closes current procedure if macro variable with same name as procedure defined
  - Defines macro variable with same name as current procedure
- So prevents code in Included procedure from being compiled twice
  - Essential for subroutines
- Better than !ifndef technique
  - One macro line does what requires two with !ifdef
  - Avoids scan of entire procedure
- Can be followed by macro variable to be used instead of procedure name



If You're Whacked About Compile Efficiency

```
PROCEDURE SUBROUTINE.FOO

!ifndef SUBROUTINE.DOIT
in ?&procfile include subroutine.doit
!endif

subroutine foo
...
call doit(%whatever, 22)
...
end subroutine
END PROCEDURE
```

Prevents opening of procedure if already Included



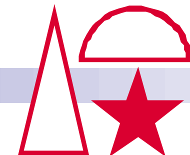
Sirius Software, Inc.

The Sirius Macro Language

- We're not really proud of it
- But it is far superior to a kick in the teeth
  - Solves some knotty little problems
    - If nothing else, !dupexit is very useful
  - Provides facilities that can be useful, if not life changing
- There **will** be more macro language enhancements in the future
  - To the current macro language
  - Or maybe a more sophisticated macro language (like User Language?)
    - Would give User Language some of the power of LISP
    - We might not be crazy enough to give this away

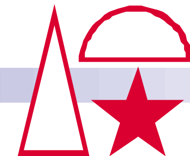
Giveaway 4 – Model 6 Support

- Adds support for arbitrary geometry (lines and columns) screens in Model 204
  - In 204 editor
  - At command line
  - In many Sirius subsystems
  - Otherwise, works like 80 character wide screen
    - Same as Model 5 support
- Uses Write Structured Field Query command to terminal
  - A way of having terminal describe itself to server
  - Enables 204 to automatically determine screen geometry
  - Works for standard models, too



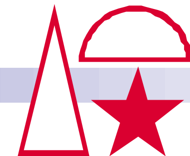
Enabling Model 6 Support

- Set SIRTERM system parameter
  - X'01' bit enables basic support
  - X'02' bit causes WSF query on every VTAM terminal connection
    - Might require VTAM definition changes?
    - Might not work with some terminals?
- Set screen geometry in 3270 emulator
  - Usually called “alternate screen size” or some such
- If SIRTERM X'02' not set, a RESET MODEL 6 sets screen size
  - Does WSF Query and determines screen geometry
  - Geometry can be changed on the fly



Why Would You Want Model 6 Support?

- Why not?
- Why limit yourself to 4 choices for screen geometry?
  - Maybe 80 columns is too little and 132 (model 5) is too much
  - Or maybe even 132 columns is not enough
- Standard screen sizes might not work well with your terminal screen size
  - Fonts are fixed number of pixels horizontally and vertically
  - Horizontal wasted space is screen width in pixels minus font width times screen width (in characters)
    - For example 1024-(132*7)=110 pixels
    - Of course, window borders enter into calculation



Conclusions

- Some nice new facilities available to (almost) all Sirius customers
- They're not life-changing, but the price is right

Any questions?