WHITEMARSH DATA MANAGEMENT SERIES

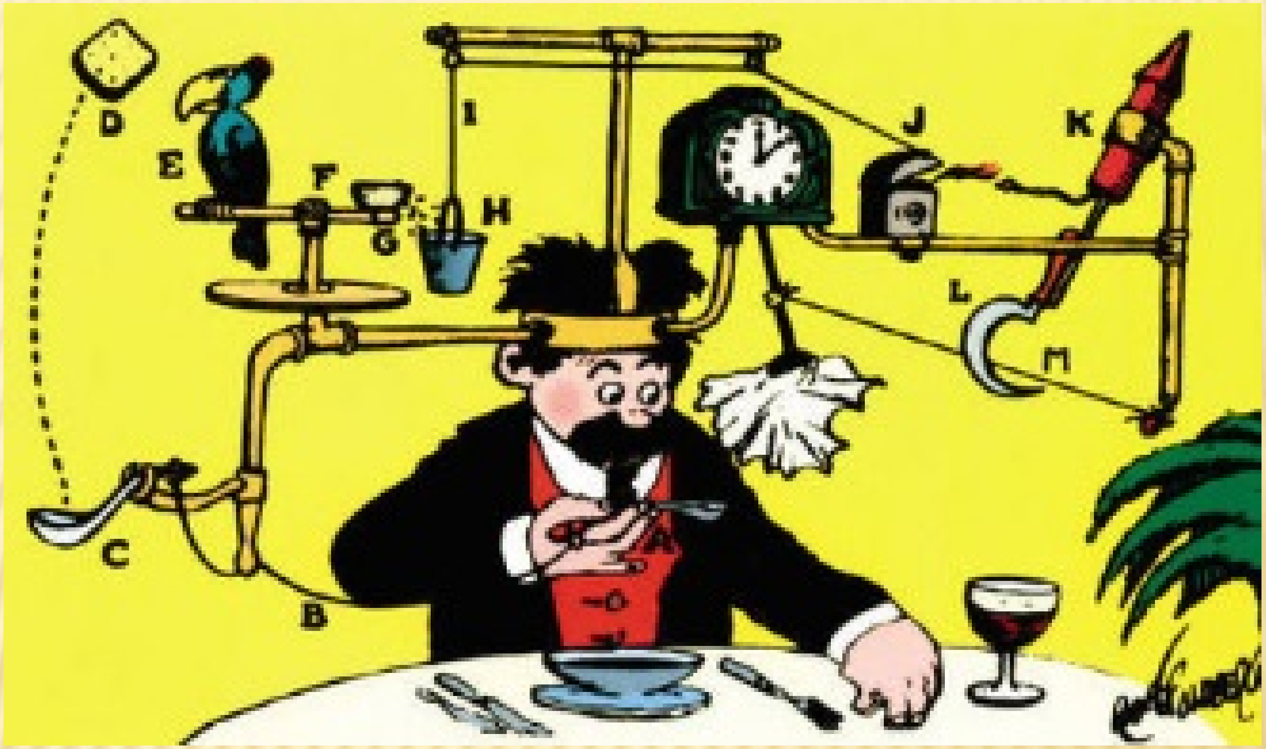
Clarion Live Presentation

Generalized
Clarion Application Software
Development
August 1, 2014

Whitemarsh Information Systems Corporation

TOPICS

SPECIALIZED VS GENERALIZED



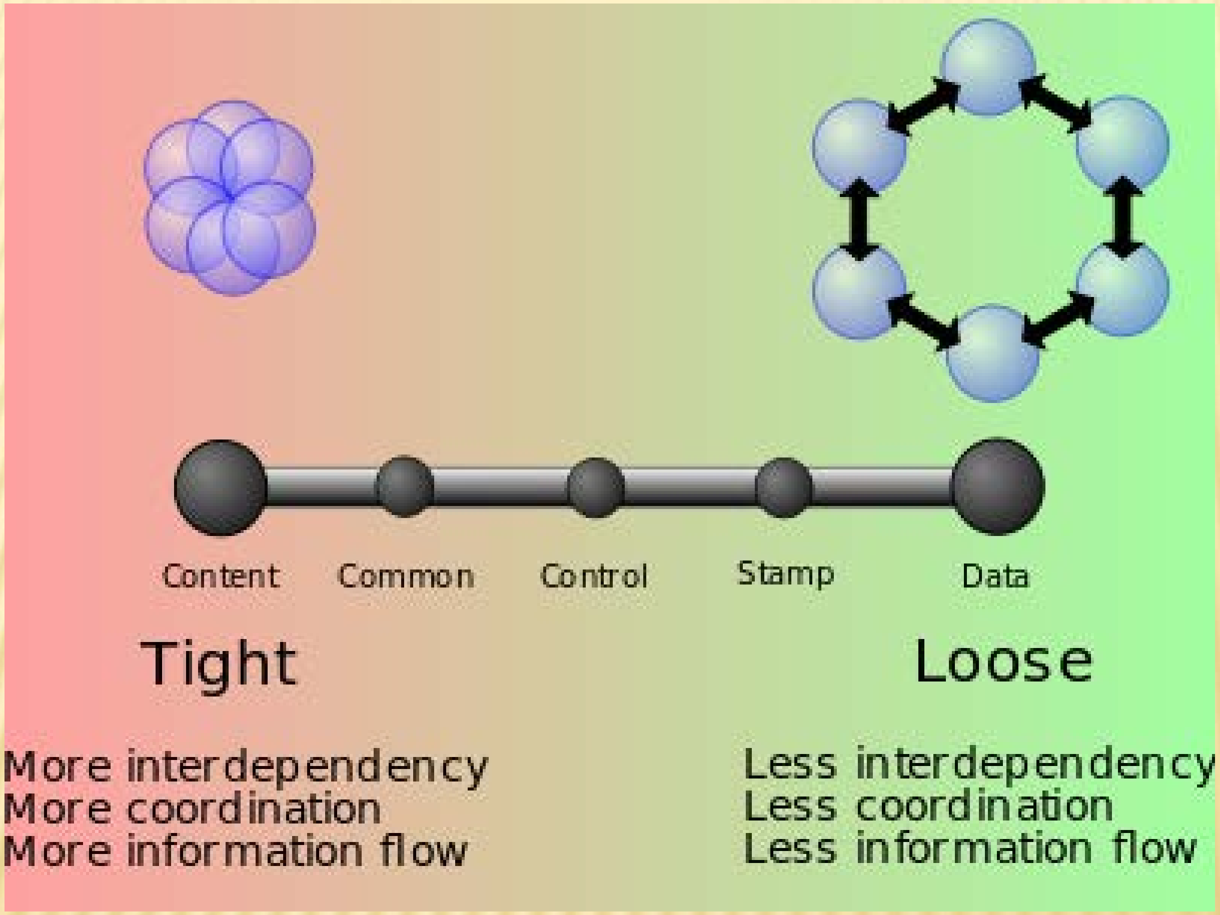
Is this Specialized or Generalized?
What is the level of Coupling and Cohesion?
Elegant architecture & Design or Hackers Paradise?

PROBLEM TO BE SOLVED: REALLOCATE "DATA" FROM ONE DATA-BASED TREE-STRUCTURE TO ANOTHER WITHOUT "LOSS, FALLING, OR BROKEN DEPENDENCIES"

BLUF (BOTTOM LINE UP FRONT)

- Coupling is critical to address between “Applications” and “Data Structures”
- Data Structure designs should be able to support many “applications”
- “Applications” should be able to support many “Data Structures”
- The Coupling between “Data Structures” and “Application” should be as loose as possible
- A Collection of Procedure Routines can be Tightly Coupled (highly cohesive) collections of 3NF contained “Procedure Routines”
- Identified Procedures that have well-defined data-based interfaces can have loose coupling
- Clarion supports Loose “Application” and “Data Structure” Coupling through Reference-Variables and “Any-Variables”
- This talk is about achieving that on a real-world practical situation: Project Management.

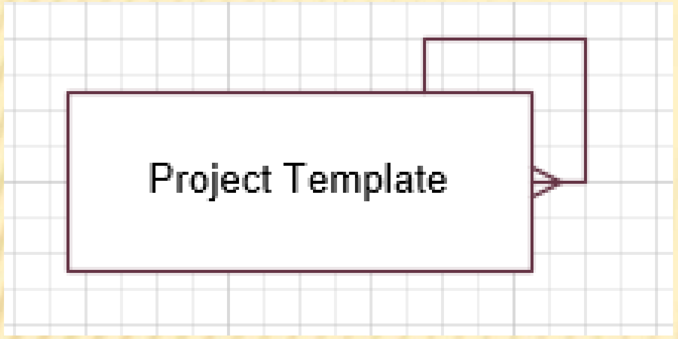
COUPLING TIGHT TO LOOSE



DEPENDENCY: CLARION AND ULTRA-TREE

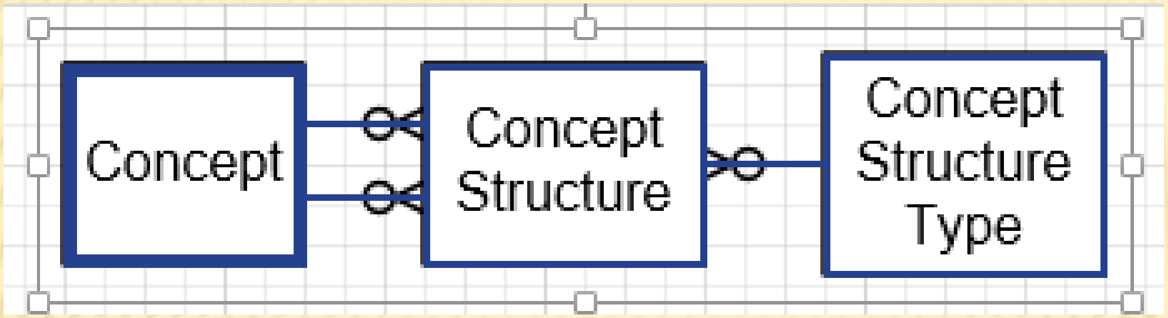
- Clarion provides the IDE and code generation environment. Mandatory that the solution exists entirely within the management of the IDE.
- UltraTree provides the fundamental Tree-Structured Data structures for:
  - Hierarchies within a single table (Recursion)
  - Networks within a three-table data structure

ULTRA TREE – HIERARCHIES



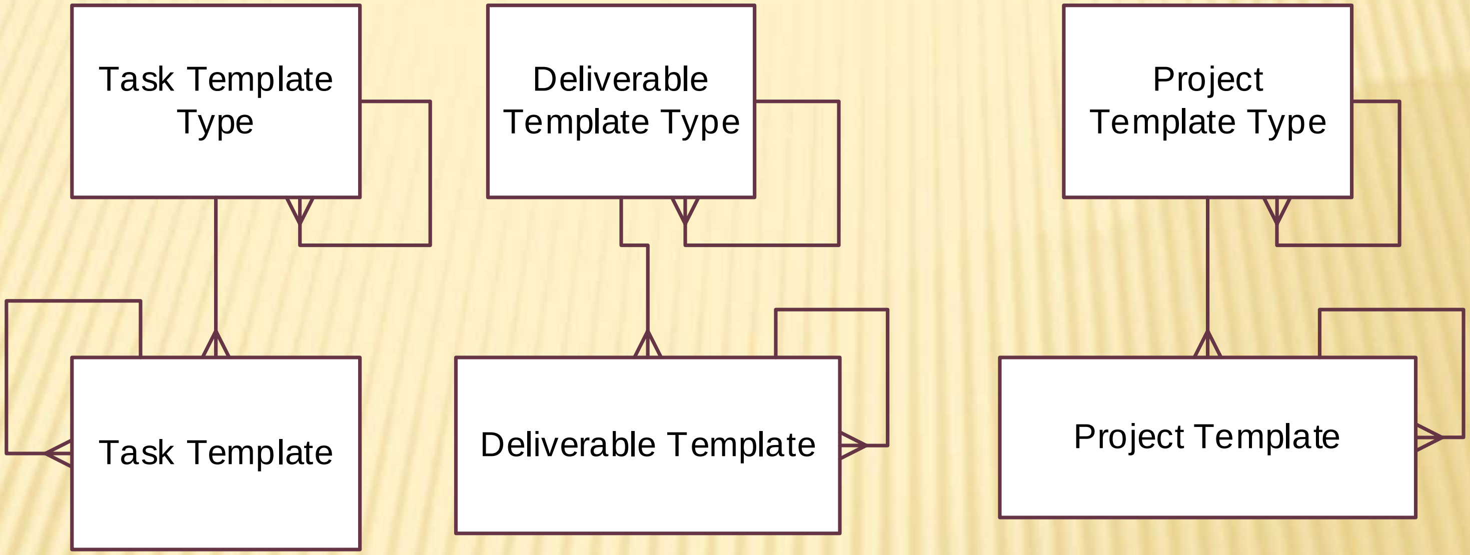
- Single Table within DCT and SQL
- UltraTree provides
  - The fundamental Tree-Structured Data structure specification
  - Hierarchical Presentation
  - Hierarchy management during updating and deletion
- Whitemarsh provides:
  - Customized Clarion Procedure Routine collections for Tree Walk (both Descending and Ascending)
  - ReAllocation of Leaf or Collections from one Hierarchy Collection to another.

ULTRA TREE – NETWORKS



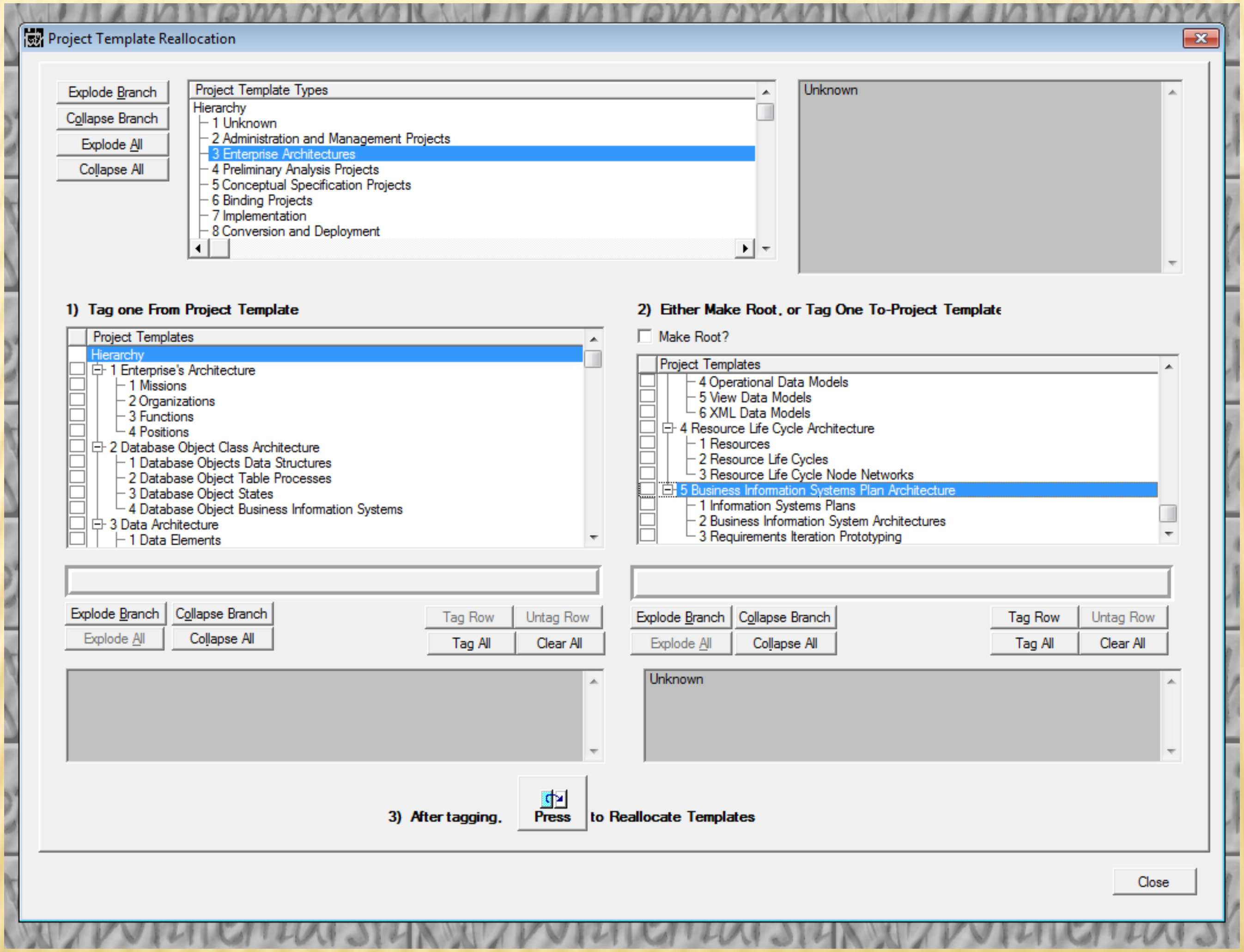
- Three Tables within DCT and SQL
- UltraTree provides
  - The fundamental Network-Structured Data structure specification. Create once, display in all relevant Hierarchy Presentations of Network
  - Hierarchy Presentation Management of Networks during updating and deletion
- Whitemarsh provides:
  - Customized Clarion Procedure Routine collections for Tree Walk (both Descending and Ascending)
  - ReAllocation is not supported as it's contrary to the fundamental nature of networks

DOMAIN OF DATA STRUCTURES



Objective: Create "software" that is flexible enough to be used in all SIX of the instances of "moving" data from within a branch of a <table name> tree to the a different branch of the same <table name> tree.

CLARION APPLICATION SCREEN-TYPED



Project Template Reallocation

Explode Branch
Collapse Branch
Explode All
Collapse All

Project Template Types
Hierarchy
- 1 Unknown
- 2 Administration and Management Projects
- 3 Enterprise Architectures
- 4 Preliminary Analysis Projects
- 5 Conceptual Specification Projects
- 6 Binding Projects
- 7 Implementation
- 8 Conversion and Deployment

Unknown

1) Tag one From Project Template

Project Templates
Hierarchy
- 1 Enterprise's Architecture
  - 1 Missions
  - 2 Organizations
  - 3 Functions
  - 4 Positions
- 2 Database Object Class Architecture
  - 1 Database Objects Data Structures
  - 2 Database Object Table Processes
  - 3 Database Object States
  - 4 Database Object Business Information Systems
- 3 Data Architecture
  - 1 Data Elements

Explode Branch | Collapse Branch | Explode All | Collapse All | Tag Row | Untag Row | Tag All | Clear All

2) Either Make Root, or Tag One To-Project Template

Make Root?

Project Templates
- 4 Operational Data Models
- 5 View Data Models
- 6 XML Data Models
- 4 Resource Life Cycle Architecture
  - 1 Resources
  - 2 Resource Life Cycles
  - 3 Resource Life Cycle Node Networks
- 5 Business Information Systems Plan Architecture
  - 1 Information Systems Plans
  - 2 Business Information System Architectures
  - 3 Requirements Iteration Prototyping

Explode Branch | Collapse Branch | Explode All | Collapse All | Tag Row | Untag Row | Tag All | Clear All

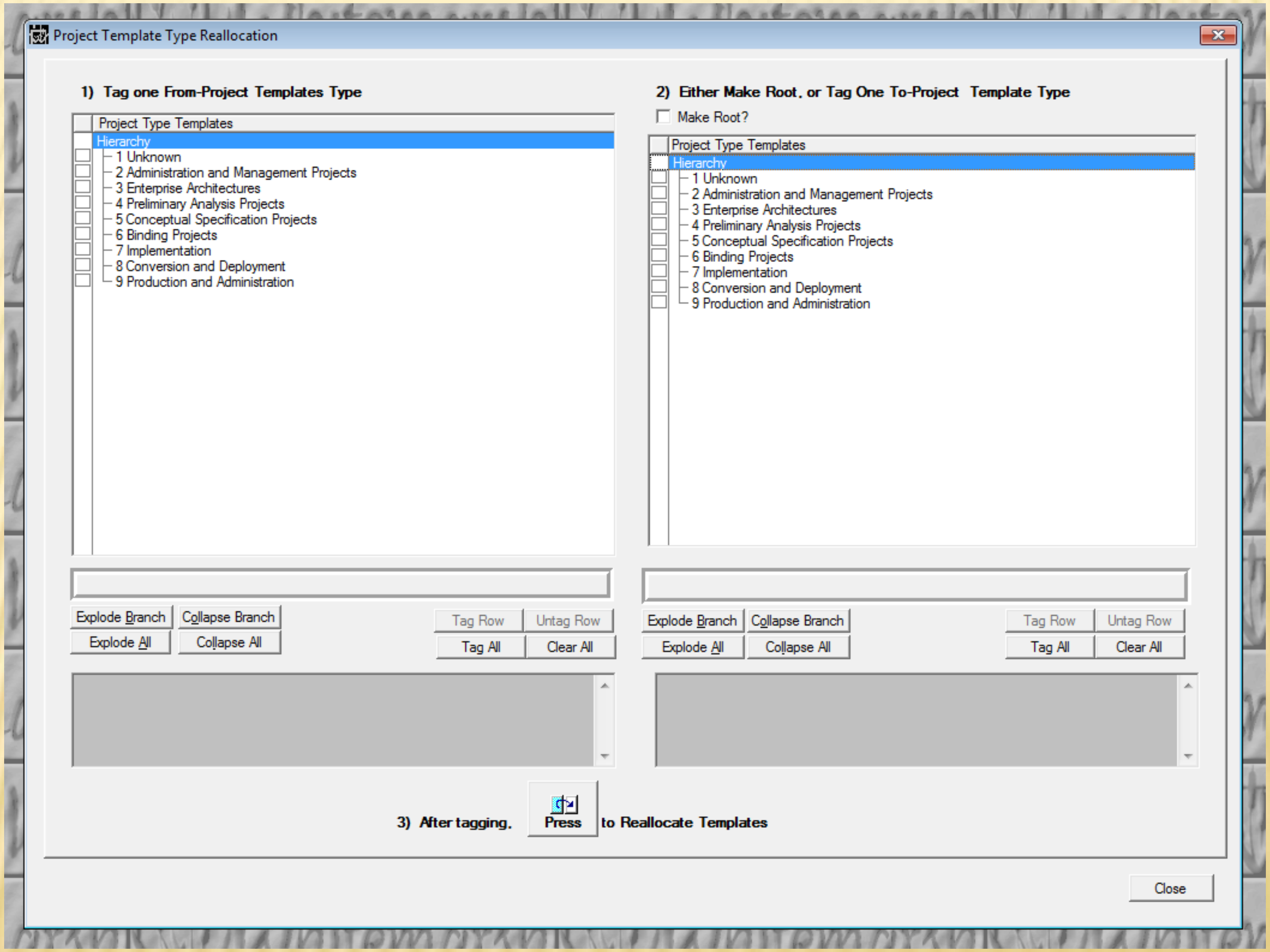
Unknown

3) After tagging, Press to Reallocate Templates

Close

Whitemarsh Information Systems Corporation

CLARION APPLICATION SCREEN-UNTYPED



Project Template Type Reallocation

1) Tag one From-Project Templates Type

Project Type Templates

- Hierarchy
  - 1 Unknown
  - 2 Administration and Management Projects
  - 3 Enterprise Architectures
  - 4 Preliminary Analysis Projects
  - 5 Conceptual Specification Projects
  - 6 Binding Projects
  - 7 Implementation
  - 8 Conversion and Deployment
  - 9 Production and Administration

2) Either Make Root, or Tag One To-Project Template Type

☐ Make Root?

Project Type Templates

- Hierarchy
  - 1 Unknown
  - 2 Administration and Management Projects
  - 3 Enterprise Architectures
  - 4 Preliminary Analysis Projects
  - 5 Conceptual Specification Projects
  - 6 Binding Projects
  - 7 Implementation
  - 8 Conversion and Deployment
  - 9 Production and Administration

3) After tagging, Press to Reallocate Templates

Whitemarsh Information Systems Corporation

SPECIALIZED SOLUTION TO THE PROBLEM

- Multiple Procedure routines with relatively high coupling and cohesion
- Fundamental process:
  - Tag a Move-From leaf or branch within the From-tree
  - Tag a Move-To leaf or branch within the To-Tree
  - Press the ReAllocate button
    - Validates that From and To are both tagged
    - Reallocates a “From and all its children” to become the child and all tagged From-Children of the “To”
    - Or, Check the Root-box to makes the From into a Root-based tree.
    - Under either scenario,
      - Traverses the From-tree to accomplishing appropriate ReNumbering and “sort key” modification within the context of the To-Tree.

SIZE OF THE SPECIALIZED CODE

- 349 lines of code (including blank lines)
- 21 discrete Procedure Routines including set of routines to manage tagging.
- 79 Specific lines of code that have one or more specialized data-based code assignments. E.g.,
  - **If** A_BusDom:BusinessDomainParentID <> **0 Then break**
  - StartParentId = A_BusDom:BusinessDomainParentID
  - **GET**(BusiDom, BUD:BusiDomPkey)

DOWN-SIDE TO A SPECIALIZED SOLUTION

- 21 discrete Procedure Routines that exist in every place where there is a ReAllocation.
- In Project Management, it's 6. Across the Metabase System, probably about 50+.
- 79 lines of code from each "copied" code set that has to be modified to bind the ReAllocate to the specific table(s).
- Tedious, Boring, and Error Prone.

APPROACH TO THE CREATION OF A GENERALIZED SOLUTION

- Fortran II had “Equates” in the early 1960s, and so it had to be somewhere in Clarion.
- Discover that Clarion has an approach for generalized coding. CW2 (1996)?
- But Clarion’s “Equates” were sort of but not really the same.
- Hunt, search, and finally after a bunch of years, discover. RefVariables and AnyVariables.

PROCESS

- Code the whole solution with specialized (data-bound) variables.
- Debug until completely correct, right, baked, "done."
- Print out all the code and "mine" for all data-binding specifications. That is,
  - Objects (tables, keys, columns, and file manager actions.
  - Columns

PROCESS (CONTINUED)

- Replace Objects with Reference Variables, and Columns with Any Variables
- Place the Reference Variables into a specific embed.
- Add all the Any Variables to the Data Pad
- Create a “MasterEquates Procedure Routine that binds the Reference and Any Variables to the appropriate Data Structure.
- Find and then substitute the Specialized Code’s objects and columns with the Reference and Any Variable Name objects and data names.

ENGINEERING AND IMPLEMENTATION OF THE GENERALIZED SOLUTION

Reference Variable:

A reference variable contains a reference to another data (its "target"). You declare a reference variable by prepending an ampersand (&) to the data type of its target.

A_SFRTab	&File
A2_SFRTab	&File
A4_SFRTab	&File
SFRTab	&File
A_SFRParentKey	&Key
A_SFRPkey	&Key
A_SFRSeqKey	&Key
A2_SFRParentKey	&Key
A2_SFRPkey	&Key
A2_SFRSeqKey	&Key
A4_SFRParentKey	&Key
A4_SFRPkey	&Key
A4_SFRSeqKey	&Key
SFRPkey	&Key
SFRSeqKey	&Key
MyFileManager	&FileManager
A_MyFileManager	&FileManager
A2_MyFileManager	&FileManager
A4_MyFileManager	&FileManager

REFERENCE VARIABLE PLACEMENT

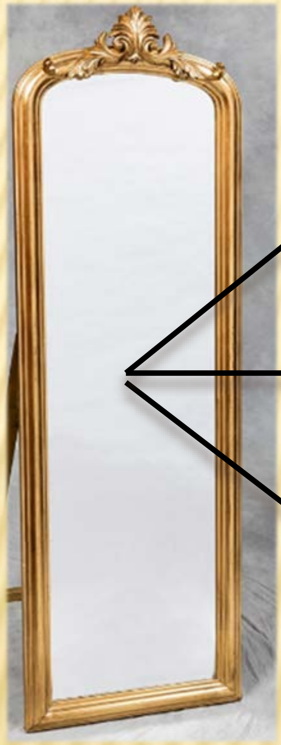
The Reference Data Variables are embedded via the IDE as follows:

Local Data →

Generated Declarations →

After Window Structure

A REFERENCE VARIABLE IS REALLY JUST A “MAGIC” MIRROR



What the Program “contains”

```
A_SFRTab            &File

Get(A_SFRTab, A_SFRPkey)
```

MasterEquates   Routine

```
A_SFRTab   &=   A_ProjectTemplate
A_SFRPkey  &=   ProjectTemplatePkey
```

What the Program “actually does”

```
Get(A_ProjectTemplate, ProjectTemplatePkey)
```

REFERENCE VARIABLE PLACEMENT

The Reference Data Variables are embedded via the IDE as follows:

Local Data →

Generated Declarations →

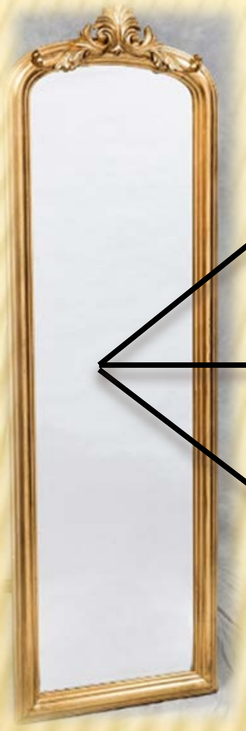
After Window Structure

ANY VARIABLES

An ANY variable is one that may contain any value (numeric or string) or a reference to any simple data type.

SFRId	ANY
SFRParentId	ANY
SFRSeq	ANY
SFRsortstring	ANY
SFRTabName	ANY
SFRTId	ANY
SFRTypeId	ANY
A_SFRId	ANY
A_SFRParentId	ANY
A_SFRSeq	ANY
A_SFRsortstring	ANY
A_SFRTabName	ANY
A_SFRTId	ANY
A2_SFRId	ANY
A2_SFRParentId	ANY
A2_SFRSeq	ANY
A2_SFRsortstring	ANY
A2_SFRTabName	ANY
A2_SFRTId	ANY
A4_SFRId	ANY
A4_SFRParentId	ANY
A4_SFRSeq	ANY
A4_SFRsortstring	ANY
A4_SFRTabName	ANY
A4_SFRTId	ANY

ANY VARIABLES ARE REALLY JUST “MAGIC” MIRRORS



What the Program “contains”

```
A_SFRId        Any
A_SFRTabName   Any
```

MasterEquates   Routine

```
A_SFRId       &=   A_Project TemplateId
A_SFRTabName  &= A_ProjectTemplateName
```

What the Program “actually does”

```
A_Project TemplateId = 784097
A_ProjectTemplateName = ‘Clarion Intro 101’
```

BUILD THE MASTER EQUATES PROCEDURE ROUTINE

```
MasterEquates       Routine
    A_SFRTab &= A_ProjectTemplate
    A_SFRParentKey &= A_ProjTempl:ProjectTemplateParentKey
    A_SFRPkey &= A_ProjTempl:ProjectTemplatePkey
    A_SFRSeqKey &= A_ProjTempl:ProjectTemplateSeqKey
    A_MyFileManager &= Access:A_ProjectTemplate

         Ditto for:       A2_<table and key Stuff>
                          A4_<table and key Stuff>
                          <table and key Stuff>
  !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    SFRId &= ProjTempl:ProjectTemplateId
    SFRParentId &= ProjTempl:ProjectTemplateParentId
    SFRSeq &= ProjTempl:ProjectTemplateSeq
    SFRTId &= ProjTempl:ProjectTemplateTypeId
    SFRSortString &= ProjTempl:sortstring
    SFRTabName &= ProjTempl:ProjectTemplateName
 !~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    Ditto for A_SFRID et al
    Ditto for A2_SFRId et al
    Ditto for A4_SFRId et al
```

SUBSTITUTE ALL THE PROCEDURE DIVISION CODE

```
MakeRoot          Routine
   StartId = A_SFRId
   NewParentID = 0
   Do GetMaxSeqNum
   primepadLV = maxseqnumLV
   Do PrimeandPad
   A_SFRSeq = maxseqnumLV
   A_SFRParentId = NewParentID
   A_SFRSortString = CLIP(LEFT(primepadLV))
   If A_MyFileManager.UPDATE()<>Level:Benign then
      Message('Could not make root. Resetting...')
      MakeRootYN = 'N'  !Resetting MakeRoot Flag
      Exit
   Else
      MakeRootYN = 'N'  !Resetting MakeRoot Flag
      StartId = A_SFRId
      Do SequenceSFR
   End
```

```
MakeRoot    Routine
   StartId = BUD:BUSIDOMID
   NewParentID = 0
   Do GetMaxSeqNum
   primepadLV = maxseqnumLV
   Do PrimeandPad
   A_BusDom:BusiDomSeq = (maxseqnumLV
   A_BusDom:sortstring = CLIP(LEFT(primepadLV))
   A_BusDom:BusinessDomainParentId = NewParentId
   If Access:A_BusDom.UPDATE()<>Level:Benign then
     Message('Could not make root.  Resetting...')
      MakeRootYN = 'N'  !Resetting MakeRoot Flag
      Exit
   Else
      MakeRootYN = 'N'  !Resetting MakeRoot Flag
      StartId = A_BuisDomId
      Do SequenceSFR
   End
```

SUMMARY

- Identify where Same-code is to be used many different places bound to different data structures
- Create solution for one and test, test, test, and once more, test.
- Identify all database objects (tables, keys and access) and make corresponding Reference Variables
- Identify all table columns and make corresponding Any Variables
- Install the Reference Variables in a Local Objects Windows Structure Embed
- Install the Any Variables into the Data Pad
- Create a Master Equates procedure Routine and map all Reference and Any Variables to data structure.
- Install the "Do MasterEquates" into the Open Window embed.
- Change out all the specialized data-based code statements with generalized data-based code statements.
- Pray for a loving and good God of Infinite Divine Providence.
- Run the thing.

FOLLOW-ON ACTIVITIES

- All this works fine.
- However, while I have one generalized set of code for all the 21 discrete Procedure Routines,
- I now have this generalized code in SIX different procedures.
- So, next step is to create a Metabase Common Code DLL and figure out how to get that working.
- Things I do not YET know:
  - Can I get this working in a generalize way with Ultra Tree (I am thinking, no!)
  - Can I re-arrange **one** of the generalized routines into two parts so that there can be a Do SpecializedProcess back to the main app with an ending Statement, which, by definition returns to the next statement after the Do SpecializedProcess (I am thinking, yes)
- Could all this have been done with Classes? I do not know. Or, if yes, would it improve readability, maintenance, and performance?

BLUF (BOTTOM LINE UP FRONT) — AGAIN

- Coupling is critical to address between "Applications" and "Data Structures"
- Data Structure designs should be able to support many "applications"
- "Applications" should be able to support many "Data Structures"
- The Coupling between "Data Structures" and "Application" should be as loose as possible
- A Collection of Procedure Routines can be Tightly Coupled (highly cohesive) collections of 3NF contained "Procedure Routines"
- Identified Procedures that have well-defined data-based interfaces can have loose coupling
- Clarion supports Loose "Application" and "Data Structure" Coupling through Reference-Variables and "Any-Variables"
- This talk showed how this can be done

QUESTIONS FOR THE GREAT GRAND PA?