

# **Information Tree Extensions within ICA (IC Objects as an Implementation of an Information Tree in Information-Centric Architecture)**

## **Overview**

The *Information Tree Structure* at the heart of *Information-Centric Architecture (ICA)* is a Data Tree with a number of critical extensions to add meaning and capture the information necessary to generate rigorous business applications from the data within those extensions. In ICA, the Information Tree structure is implemented through Information-Centric Objects (IC Objects), which define and automate the processing of ALL data in ICA business applications.

Before reviewing those extensions below, it is important to understand three foundational characteristics of IC Objects which are critical to the success of those extensions in delivering the huge gains in productivity, quality and flexibility claimed for ICA.

1. The information tree structure provides a visual image of the relationships between data components. In complex business applications, it is the relationships between pieces of information that are the most difficult to comprehend and document. The information tree structure provides a visual representation of those relationships within the various contexts of an application. An instance of a particular information tree is considered an “object” and its definition is called a “Logical Object Definition (LOD).”
2. The information tree structure supports ALL kinds of data within a business application, regardless of its source. Thus, the definition of business rules and the presentation/modification of data on human interfaces is exactly the same for all forms of data, introducing a consistency and simplicity to the whole development process.
3. The information tree structure is consistently used throughout all phases of business application development. From the earliest stages of analysis to the final stages of maintenance and debugging, that same visual structure is used without any need for its transformation from one phase to another. Thus, when a developer is debugging a business application generated through ICA, they use a visual display tool to present object content in the same structure as specified in the IC Object Definition that was initially created during analysis and then continually modified during the development/validation process.

The most important data tree extensions of ICA follow below. It is the power of these extensions that set the foundation for ICA and enable the applications of the future to be

assembled from reusable library components. They are listed as follows, with explanations of these extensions making up the body of this document.

### **Information Tree Extensions within Information-Centric Architecture**

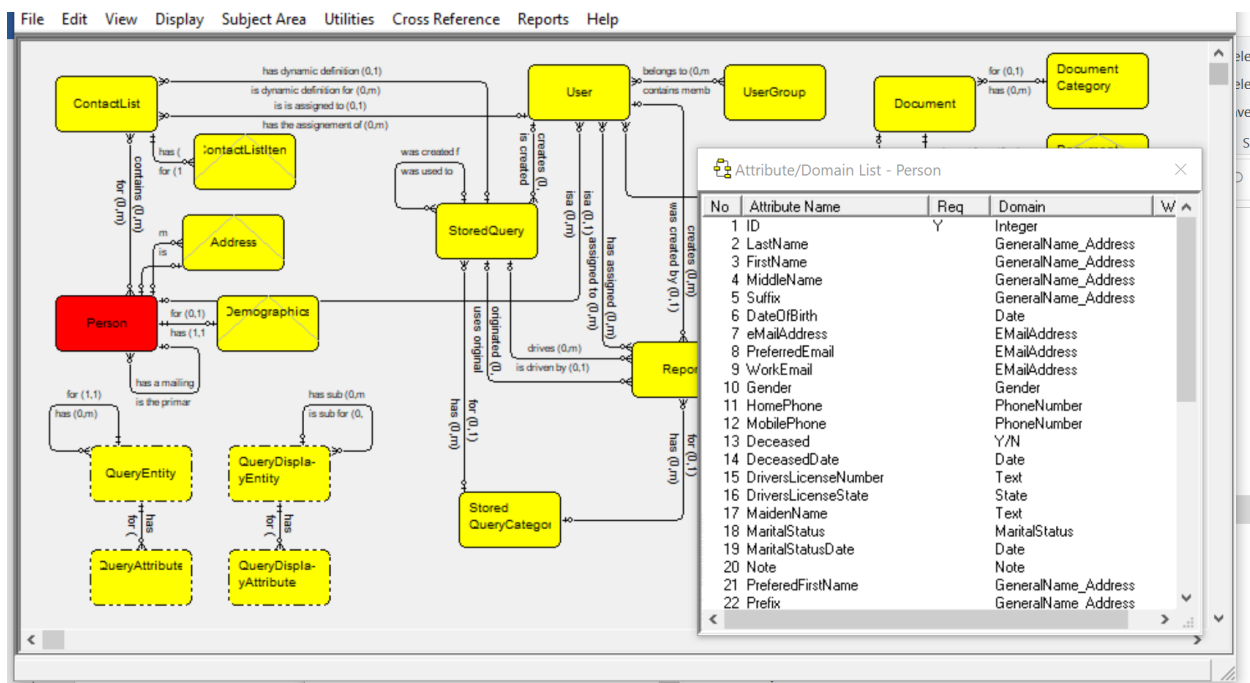
- 1. Persistent Source and Target Variations of Information Tree Data in IC Objects**
- 2. Derived Attributes**
- 3. Duplicate Entities Down Different Paths**
- 4. Includable Subobjects as Parts of an Information Tree**
- 5. IC Object Support of Work-In-Progress Processing**
- 6. Recursive Subobject Processing**
- 7. Data Integrity and Object Validation**
- 8. Cascading Delete Rules**
- 9. Business Processing Rules**
- 10. Human Interface Mapping**
- 11. Server-to-Server Communication**
- 12. Merging Components**

## I. Source and Target Variations of Information Tree Data in IC Objects

The data within an IC Object can come from a variety of sources as defined below and can be written back to the same source or used to create data in a different external form.

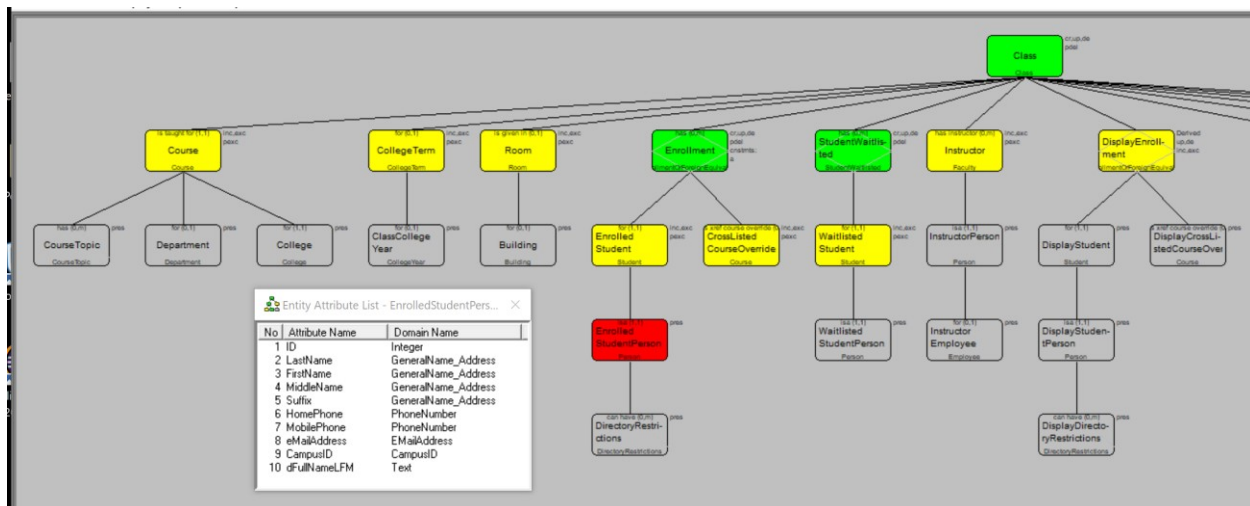
**Relational Database:** The most common source of an IC Object is a relational database, where the entities making up the IC Object are from the ER Data Model, which defines the logical form of the database. The hierarchal relationships in the tree are relationships (implemented as foreign keys in the generated physical database) from that data model. No SQL statements are ever written by the developer but are generated by the single Activate and Commit statements that are used to read and write IC Objects from and to the database.

Sample ER Data Model: The sample below shows part of the ER Data Model for the CRMBase Component Library, displaying the list of attributes for the highlighted entity, Person. Note that each attribute is assigned a *Domain*, which is an extension of the datatype for a normal data model. This extension includes the datatype of the generated database column, but also defines all internal and external values the attribute can take, which automatically validates data and automates the formatting of external data values on human interfaces. It should also be noted that an ER Data Model can be created for an existing legacy database, sometimes by importing the physical definition, allowing legacy applications to be extended and assisting their transition to current technologies.



**Relational Database IC Object:** A database IC Object definition is made up of entities, relationships and attributes from an ER Data Model. Because the ER Data Model is tied to a physical data model (which defines foreign keys for relationships), the IC Object definition has all the information necessary to generate read and write SQL and maintain the integrity of its data.

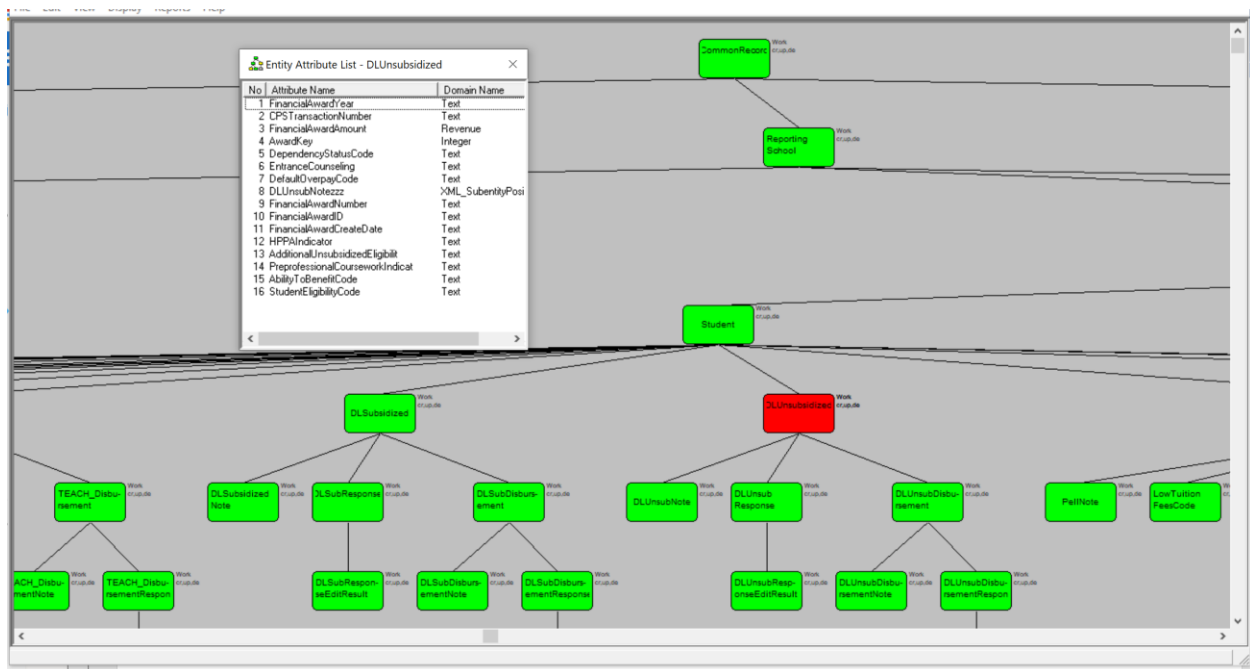
**Sample IC Object from Student Information System Application:** The sample below shows part of an IC Object to update class data in a Student Information System library. Explanations for maintaining data integrity and the significance of entity colors in the tree are provided in later sections of this document. The list of attributes are shown for the selected EnrolledStudentPerson entity, along with their assigned Domains.



[illegible]

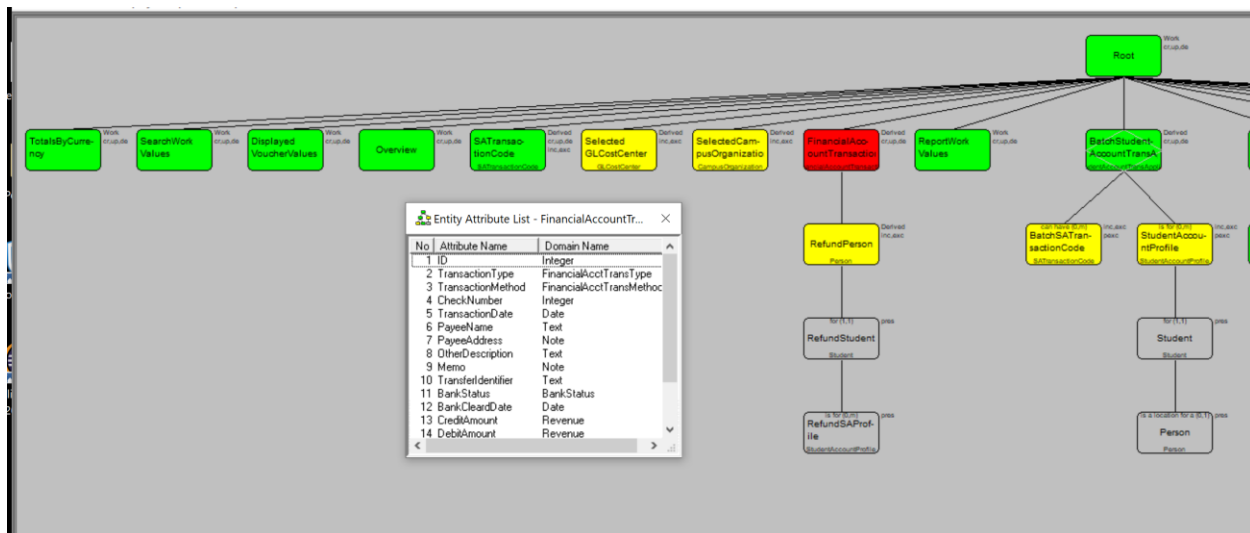
**External Data Formats:** The source of an IC Object can be any one of multiple external data formats/files, such as XML files, JSON work areas, CSV files or fixed-format files. A single Activate statement is usually all that is necessary to read the data into memory as the extension definitions to the information tree usually specify the data necessary for the Activate. Fixed-format files require a special operation to define the position of each attribute value in the record. Regardless of the source of the data in such an IC Object, the data can be written out to a format different from the source with a single Commit statement.

**Sample IC Object Definition for an XML File Structure:** The sample below shows part of an IC Object definition of a COD XML file for a financial aid application received from the federal government. The attribute list is shown for the selected DLUnsubsidized entity, along with their assigned Domains. (The DLUnsubsidized subobject contains data defining application for an unsubsidized loan.) All the detail and complexity contained in such an XML file is very difficult to understand and process without the visual aid and processing functions of an information tree.

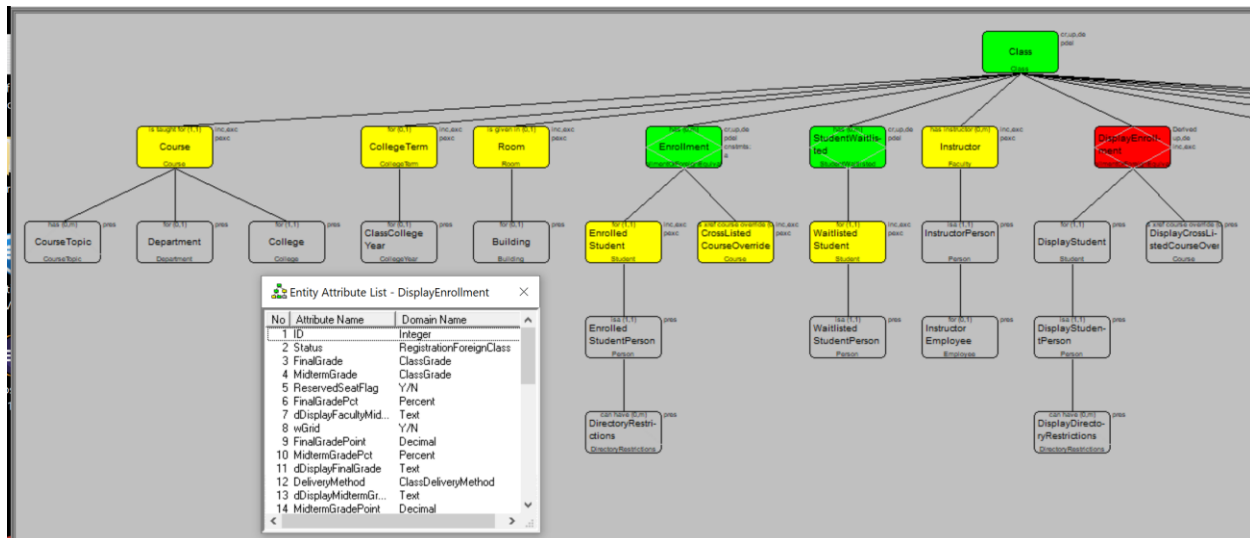


**Temporary Work Data:** An IC Object can be made up entirely of temporary work entities and relationships or temporary work components can be added to any IC Object activated from a database or an external source. In fact, almost all complex IC Objects from a database have work components (called subobjects) that are created from process rules and add important meaning to the object.

**Sample Work IC Object Definition:** The sample below shows part of an IC Object where the entities are never read or written from any database or file, but are only maintained as temporary work-in-progress data. Though most of the components are simple work entities, there are entities defined in the ER Data Model, such as the highlighted entity, **FinancialAccountTransaction** and its children **RefundPerson** and **RefundPersonStudent**. They contain temporary work data for database data that is being processed but not written.



Sample Work IC Object Definition: The sample below shows part of the IC Object displayed earlier of updatable class data in a Student Information System library. Though most of the entities shown are read/written to/from the database, the selected DisplayEnrollment subobject is temporary work data used for displaying selected Enrollment data.



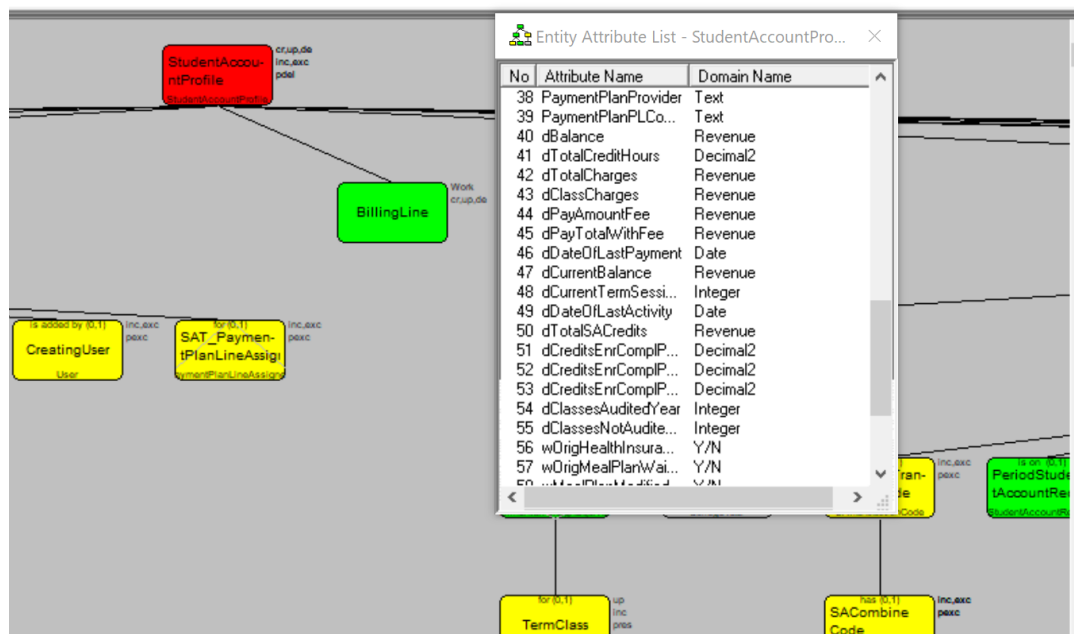


## II. Derived Attributes

A particularly valuable extension to the information tree is the concept of a derived attribute, which is an attribute defined on an entity in an IC Object through a Process Rule. What makes derived attributes so powerful is that they are visualized and utilized as just another attribute in the object, whether presented on an interface or used in a processing rule. It is another example of the IC objects introducing consistency in data/information handling throughout the system.

The rules for defining a derived attribute can be either procedural or nonprocedural, as explained in the section, *Business Processing Rules*, later in this document.

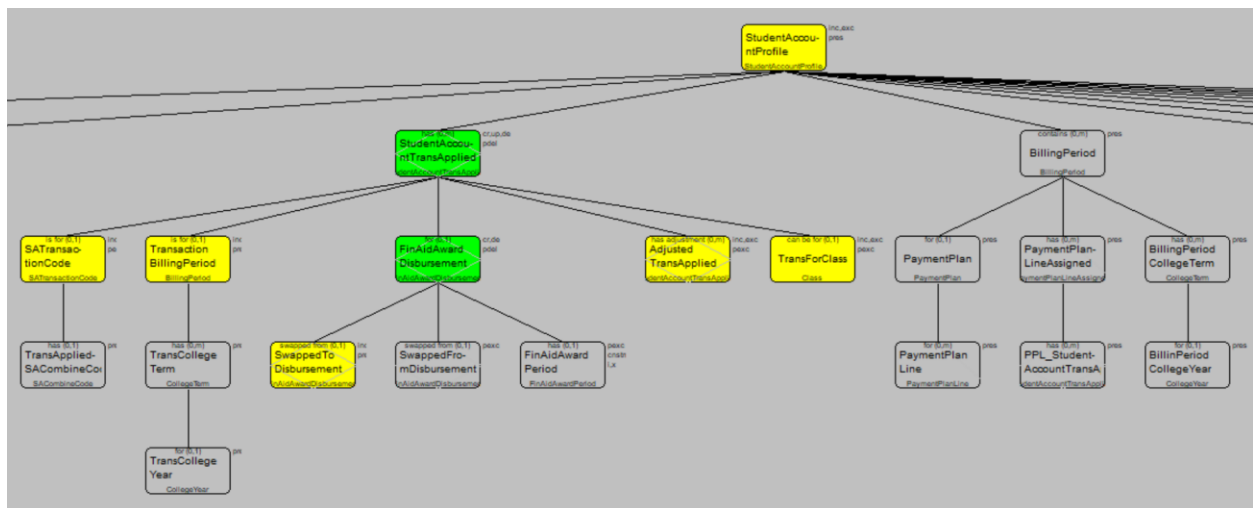
Sample Attribute List Containing a Number of Derived Attributes: Note that it is the convention of this application to prefix all derived attributes with the lower-case “d” and all work attributes with the lower-case “w”. This is simply a common convention, however, and has no actual significance in derived attribute processing.



### III. Duplicate Entities Down Different Paths

One important variation of an information tree from a regular data tree is that the same instance of an entity can occur in one IC Object down different relationship paths. Though not as common as derived attributes, it is critical when needed, because some information cannot be fully defined except by containing more than one relationship to the same entity instance within the same object.

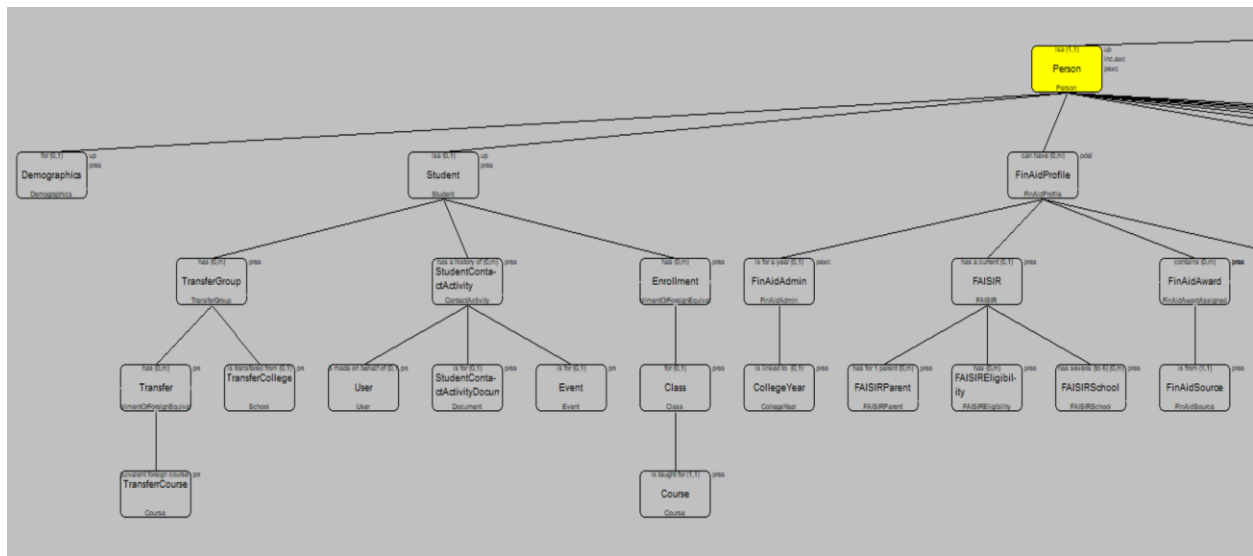
Sample Billing IC Object: An example occurs below in an IC Object used for generating billing transactions in the Student Information System application. The root entity of the object is StudentAccountProfile, which anchors all the billing data for a student. Part of the data necessary for billing (but maintained through another object) is the BillingPeriod subobject, which identifies Payment Plan information for that student and college term. When a billing rule generates a billing transaction (StudentAccountTransApplied subobject in the object below), that transaction must be tied to the correct BillingPeriod. This is accomplished by including the current BillingPeriod entity in the object as the TransactionBillingPeriod, which pulls along the BillingPeriod CollegeTerm and CollegeYear data with it. Though the data in the entities is the same, the meaning is slightly different depending on the path. The data down the BillingPeriod path identifies period data that is currently being processing during billing. The data down the StudentAccountTransApplied path identifies the period data for which the transaction was identified. Both are necessary for proper understanding and processing.



## IV. Includable Subobjects as Parts of an Information Tree

One nonintuitive capability of an information tree is the definition of subobjects (subtree structures) as part of an object that can be “included” from a similar subobject in either a separate object or in another path in the same object. This allows object data (represented as a subobject, the target) to be built in memory (using the work-in-progress capability of IC Objects) from data (a subobject, the source) in either the same or another object using a single “Include” command within a processing rule or from an include function on an interface, such as a combo box. Actually, combo boxes are often used in this manner, where the source of the include is represented in the control as the drop-down list of an identifying value in the root of the subobject and the target of the include is the same identifying value of the included subobject. Note that the identifying values in both the source and the target can be derived attributes.

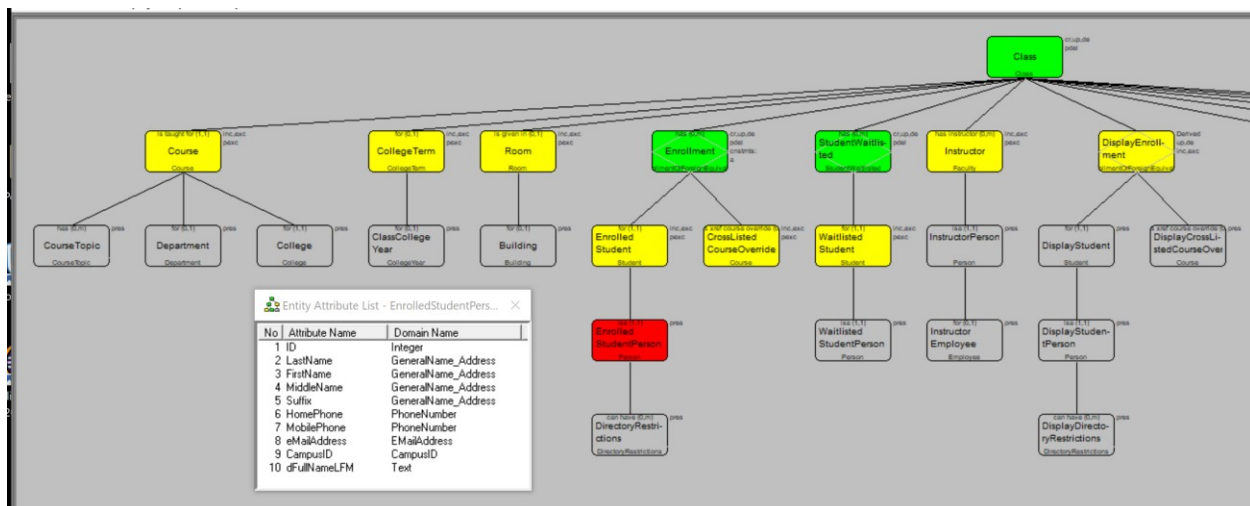
Sample IC Object Showing Included Person Subobject: The example below is for a Prospect maintenance object, where a new prospect record for a former student is created by including the Person subobject from another IC Object containing that information. A single Include from either a processing statement or an interface control brings over all the data from the source object to the target object, populating all the data in the modified IC Object, which can then be displayed directly on human interfaces or used in processing rules.



## V. IC Object Support of Work-In-Progress Processing

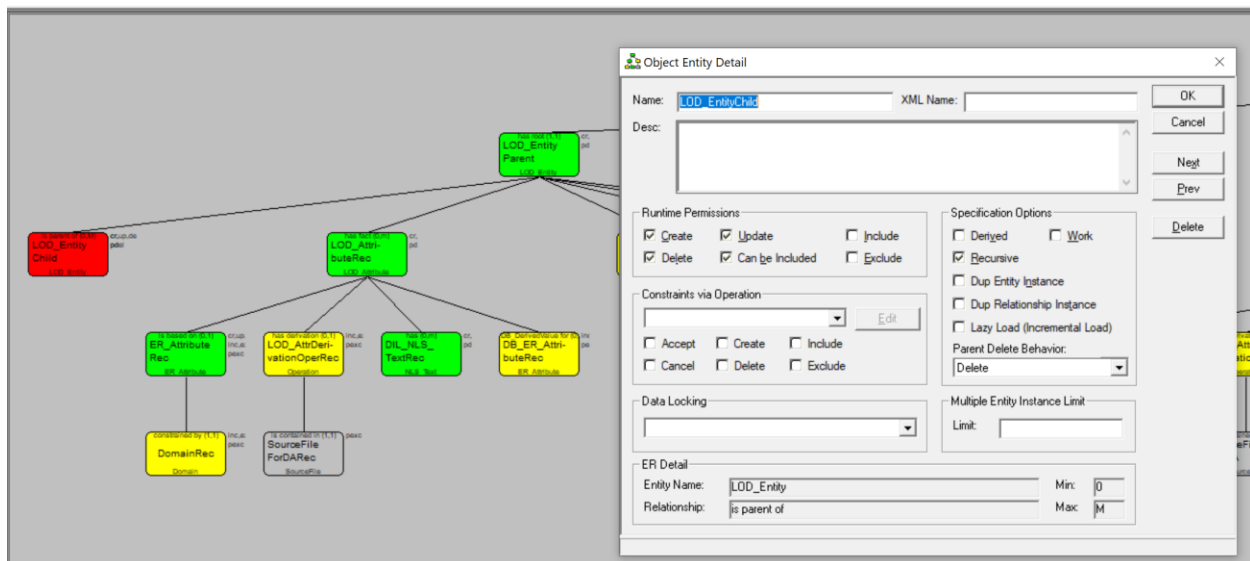
A critical function of all maintenance applications and an important part of many complex business processes is the automation and management of work-in-progress data (sometimes called support for the “long-business transaction problem”). Regardless of the source of data that needs to be processed and/or modified, work-in-progress processing supports the management of data in memory, freeing the application from having to keep track of modified data through its own processing rules. IC Objects do all of this automatically so that application processing rules or human interfaces need only add, delete or update entity components. At the end of the process/transaction, the application rules need only either “cancel” the work-in-progress object or “commit” it back to its source, either function accomplished with a single statement.

Sample Work IC Object Definition: The sample class maintenance object shown earlier and below is an example of a work-in-progress object. Either through processing rules or an interactive set of update web pages, Class, Enrollment and StudentWaitlisted entries are created and Course, CollegeTerm, Room and Instructor subobjects are included from other IC Objects. At the end of the series of interactions, the application user specifies a function (ie., strikes a button) that either cancels the object (dropping it from memory) or saves the object, executing the necessary SQL calls to add, delete and update to the database all changes made to the work-in-progress object instance.

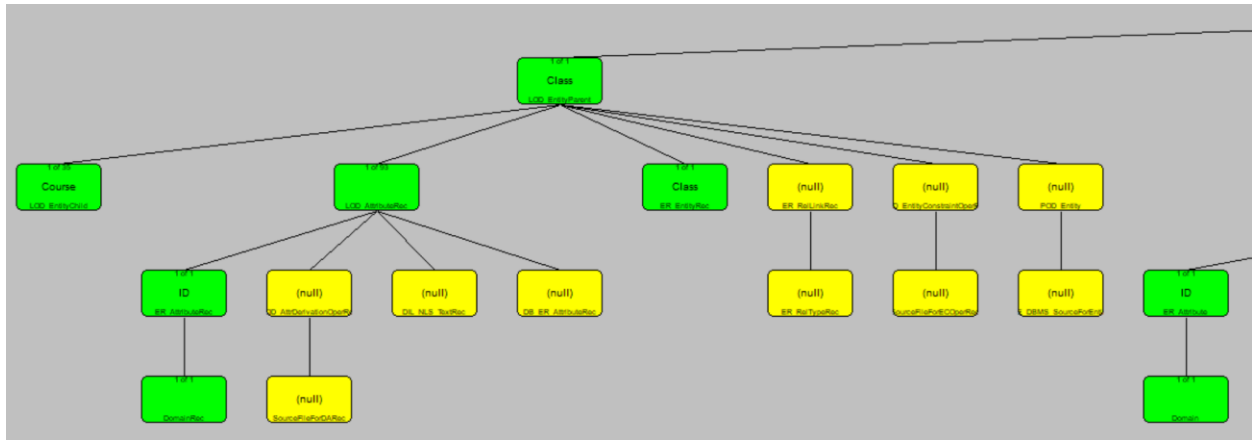


## VI. Recursive Subobject Processing

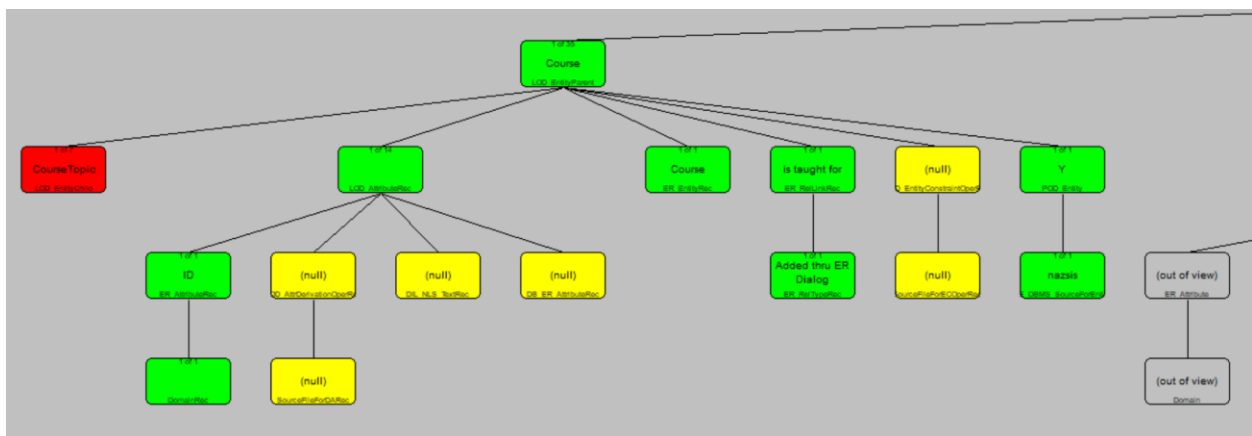
A fairly rare data requirement, but one that is critical when needed, is the recursive processing of data through the same relationship. Actually, it plays a very important role in ICA because some of the meta objects (particularly the IC Object definition for IC Objects) require it for navigating the tree structure of information trees. The first sample below shows the recursive subobject structure used in the definition of an IC Object. Note that the two entities, “LOD\_EntityParent” and “LOD\_EntityChild” are for the same entity, “LOD\_Entity” in the data model and that the detail for the highlighted LOD\_EntityChild entry has the checkbox titled “Recursive” selected. During execution, the recursive structure is navigated through the SetViewToSubobject( View, “LOD\_EntityChild” ) operation, which continues to step down and up through the recursive relationship. The second and third samples below show a run-time display of such an object using the tool’s Object Browser. The first of the two shows the object before stepping down the recursive relationship and the second after stepping back up.



Sample Object Browser Display of Recursive Relationship Before Stepping Down: Note that the LOD\_EntityParent entity instance is for “Class” and the LOD\_EntityChild entity instance is for “Course”. Also, note that other entity instances in the object are visible in the lower right corner of the display.



Sample Object Browser Display of Recursive Relationship After Stepping Down: Note that the LOD\_EntityParent entity instance is for “Course” (the LOD\_EntityChild entry in the previous display) and the LOD\_EntityChild entity instance is for “CourseTopic”, one step down the recursive relationship. Also, note that other entity instances in the object are NOT visible in the lower right corner of the display as we’re stepping only through the recursive structure.

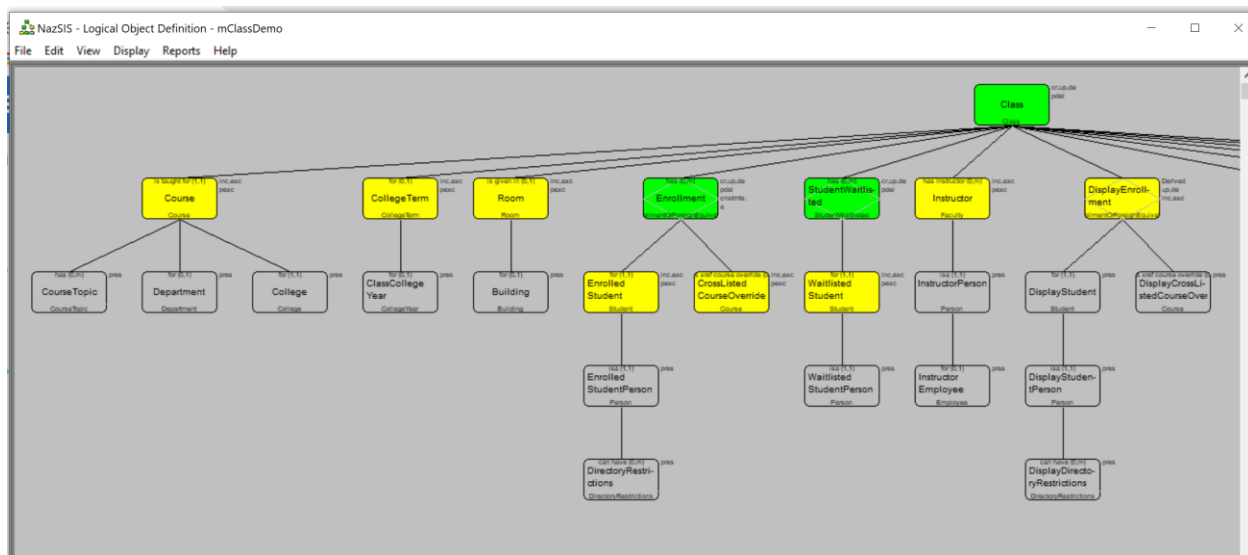


## VII. Data Integrity and Object Validation

Managing data integrity through the IC Object definition is critical both because ensuring data integrity is crucial to business application processing and the programming tasks normally required to assure such integrity are often time-consuming, obscure and incomplete. IC Objects assist, document and automate that task in three ways:

1. As defined earlier in this document, a Domain is assigned to each attribute in an IC Object, which specifies (either through a list or algorithm) all the internal and external values that an attribute can be assigned. It is thus impossible to assign a value to an attribute that violates the Domain rule.
2. As defined later in this document under *Business Processing Rules*, procedural or nonprocedural rules can be defined to validate complex interactions between object data components.
3. *Runtime Permissions* are assigned to each entity in an IC Object to control what database maintenance actions are allowed to maintain data through that object. The basic permissions are read, create, update, delete and include. To assist in documentation and understanding, colors are assigned to each of the three most common conditions in the visual display: green for create/delete, yellow for include and gray for read-only. This allows anyone reviewing the documentation to get a quick overview on how the object is used relative to the application as a whole.

Class IC Object Display Shown Earlier in this Document: Note that only 3 entities can be created for the object, with most data being pulled in by including data from other objects.



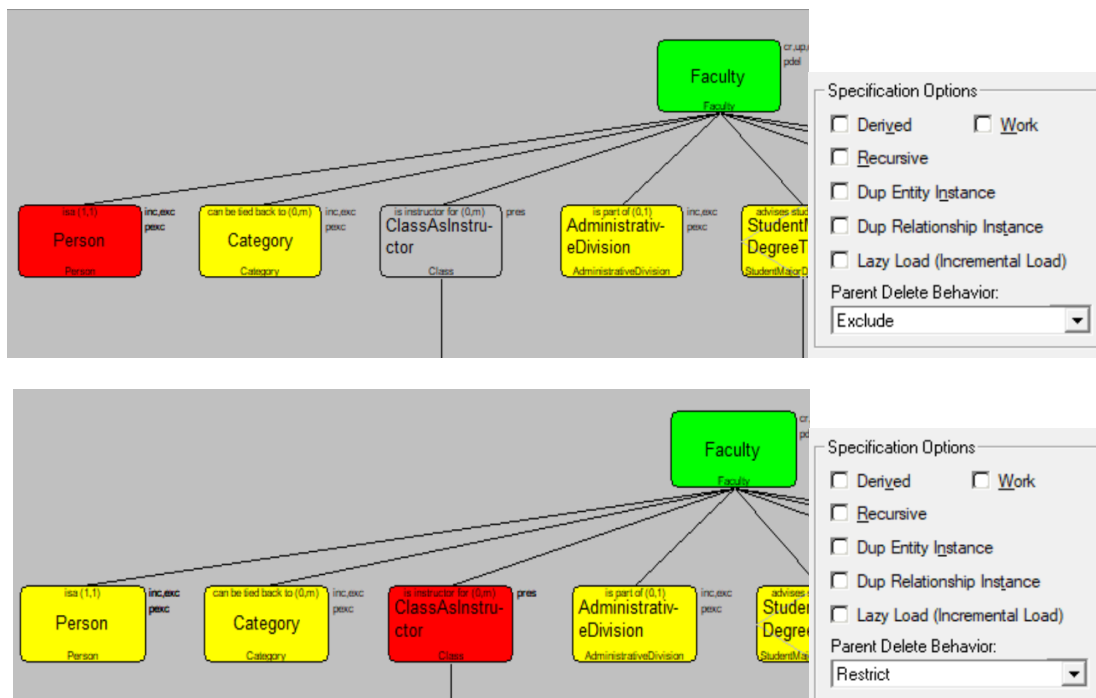
## VIII. Cascading Delete Rules

Runtime permissions also include what is called Parent Delete Behavior, which is necessary to control and automate the deleting of related data within a long business transaction. There are two issues:

1. The control over what entities can be deleted within an IC Object for maintaining data integrity as covered in the previous section.
2. The documentation and implementation for deleting other entities and relationships tied to an entity when that entity is deleted.

If an entity's Parent Delete Behavior is "Restrict", then a validation error will occur if the parent entity is deleted. If an entity's Parent Delete Behavior is "Delete", then that entity will be automatically deleted when its parent in the IC Object is deleted. If an entity's Parent Delete Behavior is "Exclude", then that entity will be automatically excluded (meaning its relationship will be severed) when its parent in the IC Object is deleted.

Sample Parent Delete Behavior of Exclude and Restrict: In the first sample below, the behavior of Exclude means that the relationship to the Person entity is removed if the Faculty entry is deleted. In the second sample below, the behavior of Restrict means the Faculty entry cannot be deleted if it is tied to a Class entry (ie., the faculty member is an instructor for a class).

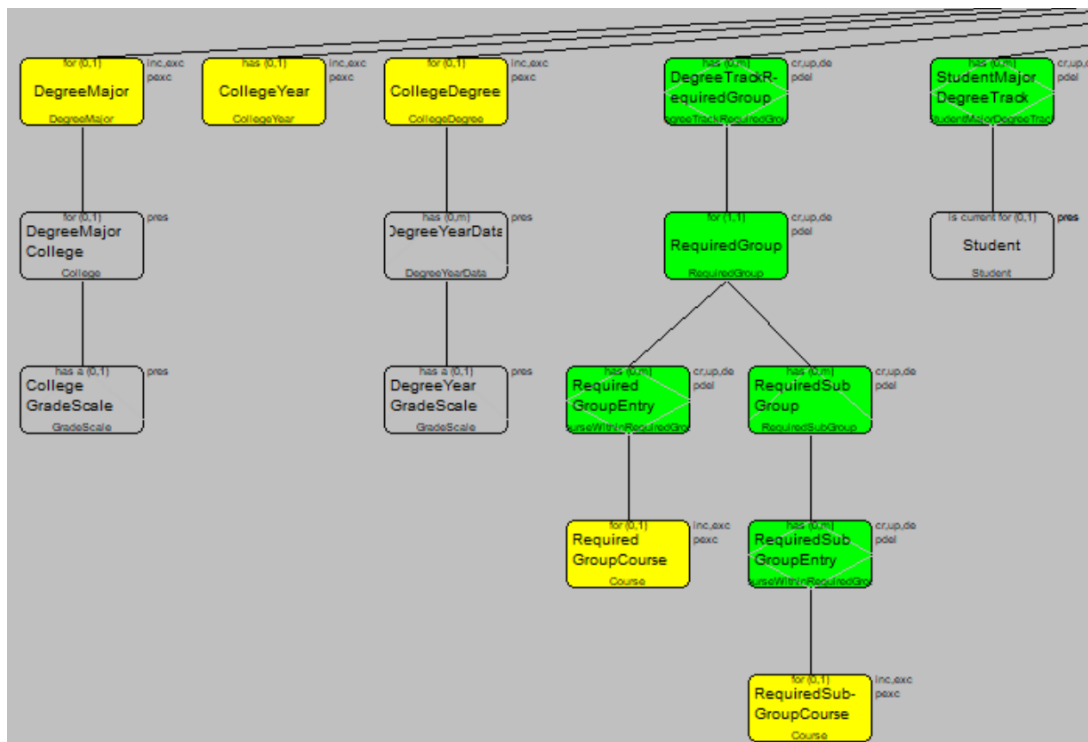




Sample Parent Delete Behavior, Including Cascading Delete: The sample below presents examples of each kind of parent delete behavior when the root entity, DegreeTrack, is deleted. When a single Delete command on the root entity, DegreeTrack, is executed, either through a procedural or nonprocedural command, all Restrict rules are validated. Thus, if a Student entity exists in the IC Object, its parent entity, StudentMajorDegreeTrack, cannot be deleted, which in turn means its parent DegreeTrack cannot be deleted and the Delete command will fail with an error code. If there is no error on the Delete command, the DegreeTrack entity in the work-in-progress object is flagged for delete.

When a Commit command is executed on the IC Object instance, the other delete rules are triggered, which in this case means the DegreeMajor, CollegeYear and CollegeDegree entities are excluded (ie., their relationships are deleted). Also, all DegreeTrackRequiredGroup entries and their children, RequiredGroup, RequiredGroupEntry, RequiredGroupSubGroup and RequiredGroupSubEntry, which have a Parent Delete Rule of Delete, will be deleted and the two children, RequiredGroupCourse and RequiredSubGroupCourse will be excluded.

The result of this auto behavior, driven by the Parent Delete Behavior of the IC Object, is much clearer documentation of the rules, the elimination of the normal programming effort to execute such rules and improved integrity resulting from the visual and more understandable rule specification.



## IX. Business Processing Rules

As has been discovered in billing interfaces used by at least one of the leading health care systems, a data tree structure lends itself to the definition of high-level business rules. That characteristic is effectively used with the information tree structures of ICDP to specify rules for a number of purposes, as shown in this section and the section that follows. Those business rules are defined in ICDP using two different techniques. The first technique specifies rules through nonprocedural definitions defined against an IC Object, as shown in the three “Nonprocedural...” examples below. The second technique utilizes procedural rules, also defined against an IC Object, as described in the “Procedural Rules” section that follows.

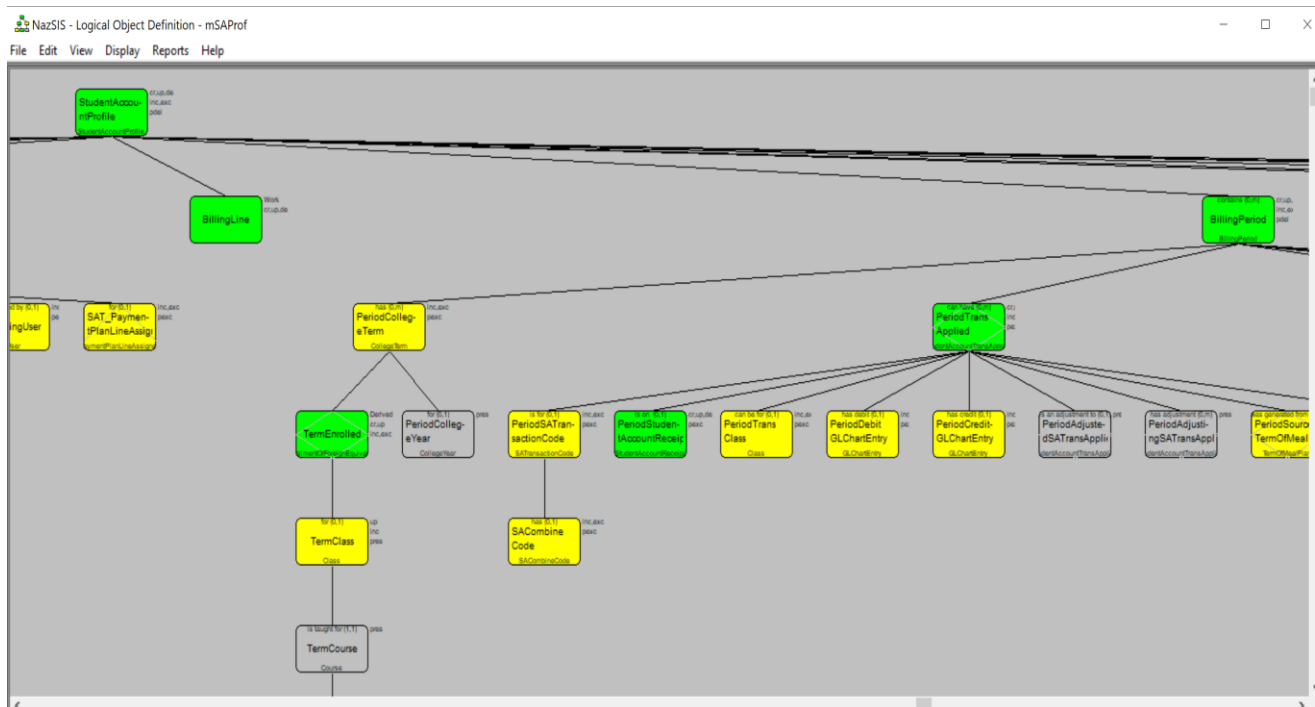
The first nonprocedural example is for the specification of a billing rule defined against the IC Object shown below under “Billing Sample IC Object.” That nonprocedural example is shown on two sample pages: The sample first page, shown below under “Nonprocedural Rule Billing Example Interface,” defines a Boolean rule and each component of the Boolean statement. The second sample page, shown below under “Nonprocedural Rule Query Example Interface,” presents a selectable version of a billing IC object containing the information that can be used within a billing rule, along with the entity/attribute rule criteria created from that object. Entries from the “Entity/Attribute” data group are selected when adding a Boolean statement to the “Criteria” data group.

The second nonprocedural example, shown below under “Nonprocedural Rule Query Example Interface,” is for a Query definition that is a part of the CRM reusable library. The interface is similar to that of billing, except that the top data group defines the data to be returned from the query and the second data group defines the Boolean rule (query criteria) which qualifies the data being returned. The third data group defines the selectable structure and data of the IC query object.

The third nonprocedural example is from the ICDP Tool Set itself and defines the rule generating a “derived attribute” for an object. Because the Tool Set runs as an MS Windows system, the layout of the window is quite different. Also, the rule structure itself is more flexible as it is comprised of a calculation set containing both a condition Boolean statement, “Rule Criteria”, and a calculation statement, “Rule Calculation.” The condition statement provides qualification for the calculation statement. Note that, though the use is different, the information tree structure and rule definition content are much the same.

## Billing Sample IC Object – SIS Student Accounts Billing

The following is part of the diagram for a rigorous IC Object for billing in a large Student Information System. The object definition (which contains more than 100 entity components) supports a visual understanding of the necessary data structures and processing rules to manage and automate a very complex billing problem. The data relationships behind such a complex problem are almost impossible to understand and automate without an effective visual image of those relationships as implemented in ICDP using the information tree structure.



## Nonprocedural Rule Billing Example Interface

As noted above, this example has two sections. The top section defines a Boolean rule and each component of the Boolean statement. The second section defines the structure and data of the IC object containing the information that can be used within a billing rule. Entries from the second section are selected to be added to the first.

StudentAccounts > Administration > Rule Set Qualification

Save & ReturnReturn

DescriptionTuition - Audits - Uses Refund Level Rules

Generate Charges for Full Year (ie., All Terms in Year)☐

Generate Charges Using Term Amounts for Student Entry Year☐

Use Special Billing Charge RuleTuition: Charge Per Credit with Refund Level

Charge Amount FactormSAProf.PeriodCollegeTerm.dClassesAudited

HelpSelectRemove

Criteria

Derived Expression (Default is AND's if no Boolean)C1

Boolean Expression (of form "(C1 OR C2) AND C3")

Parse

50

1 - 1 / 1

ConditionAttributesOperatorValueActionScopeDeleteUpdateDuplicate

C1.....PeriodCollegeTerm.dClassesAudited>0

1 - 1 / 1

Entity/Attribute

StudentAccountProfile

200

1 - 132 / 132

EntitySelect

StudentAccountProfileSelect

.....StudentSelect

.....PersonSelect

.....FinAidProfileSelect

StudentAccountProfile

200

1 - 65 / 65

AttributeAdd

IDAdd

NoteAdd

BalanceForwardAdd

AccountBalanceAdd

## Nonprocedural Rule Query Example Interface

As noted above, this second nonprocedural example is for a Query definition that is a part of the CRM reusable library. The interface is similar to that of billing, except that the top section (“Criteria”) defines the Boolean rule which qualifies the data being returned and the second section (“Display Options”) defines the data to be returned from the query. The third section defines the structure and data of the IC query object.

StudentAccounts > Reports > Query Detail

RefreshSave & ReturnReturn

NameGraduation Date Range

Category

Report Title

Description

ViewqStuErr

☐ Optimize Query (Advanced)

Open AllClose All

Criteria

ExpressionC1

50

Refresh

1 - 1 / 1

Condition	Attributes	Operator	Value	External Value	Action	Scope	Subselect Qualification	Delete	Update	Duplicate
C1	.....GraduationDate.Date	subselect			ANY	Student	>= 2020-09-01 AND <= 2021-02-05			<div>Duplicate</div>

1 - 1 / 1

Display Options

50

1 - 4 / 4

Attributes	Display	Force Read	Format Max Cardinality As 1	Column Header	Display Order	Display Length	Sort Order
...Person.CampusID	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CampusID	1		
...Person.dFullNameLFM	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Student Name	2		1
.....GraduationDate.Date	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Date			
.....GraduationDate.Year	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Year			

1 - 4 / 4

Query Extension Display Options

Entity/Attribute

50

1 - 47 / 47

1 - 050 / 64

Entities	Select	More Info	Attributes	Add
<input type="checkbox"/> Student (0-m)	<div>Select</div>	<div>More Info</div>	<input type="checkbox"/> ID	<div>Add</div>
<input type="checkbox"/> ...Person (0-1)	<div>Select</div>	<div>More Info</div>	<input type="checkbox"/> Status	<div>Add</div>
<input type="checkbox"/> .....Address (0-1)	<div>Select</div>	<div>More Info</div>	<input type="checkbox"/> CurrentLevel	<div>Add</div>
<input type="checkbox"/> .....Demographics (0-1)	<div>Select</div>	<div>More Info</div>	<input type="checkbox"/> eMailAddress	<div>Add</div>
<input type="checkbox"/> .....Prospect (0-m)	<div>Select</div>	<div>More Info</div>	<input type="checkbox"/> ResidencyStatus	<div>Add</div>
<input type="checkbox"/> .....Counselor (0-1)	<div>Select</div>	<div>More Info</div>	<input type="checkbox"/> DateOfDeparture	<div>Add</div>
<input type="checkbox"/> .....CounselorPerson (0-1)	<div>Select</div>	<div>More Info</div>	<input type="checkbox"/> FormYearMonth	<div>Add</div>

## Nonprocedural Rule Derived Attribute Interface

In the example below, the Rule Criteria, C1, identifies the repeated EnrollmentWaiverTransfer entries ("Loop") where the Rule Calculation "Sum" is applied for Status values "T" or "C".

Derived Attribute Calculation Set Update

Description: Total Credits for "T" (Enrolled) and "C" (Completed) Student Enrollment Entries

Rule Criteria: C1

Rule Calculation: C2

OK

Cancel

Rule Set Parameters

Param...	Entity / Attribute	Qual	Value	Sub Action	Scope
C1	EnrollmentWaiverTransfer.Status	Subselect	"T" OR "C"	Loop	
C2	EnrollmentWaiverTransfer.CreditHours			Sum	

Potential Rule Set Entity / Attributes

Student

- Person
  - Address
  - Demographics
  - Prospect
    - Counselor
      - CounselorPerson
      - FirstMajorChoice
      - SecondMajorChoice
      - ThirdMajorChoice
    - Church
  - AdministrativeDivision
  - EnrollmentWaiverTransfer**
  - Class
    - Course
      - College
    - CollegeTerm
      - CollegeYear
    - Faculty
      - FacultyPrimaryCampusLocation

Attribute

- ID
- DeliveryMethod
- FinalGradePct
- FinalGradePoint
- Status
- FinalGrade
- TakingClassType
- CreditHours
- DroppedDate
- GradUndergradOverrideFlag
- dCollegeType

## Procedural Business Rules

There are limitations of the nonprocedural definitions in handling very complex rules. So, ICDP utilizes the normal structure of procedural rules (ie., program structure) to define very complex algorithms. However, it is important to note that those procedural statements reference IC Objects through that same consistent information tree structure as used in the high-level rules.

Thus an "IF" statement can reference an attribute as an object/entity/attribute combination:

```
IF mPerson.Person.LastName = "Smith"
```

```
IF mPerson.Person.LastName = IContactList.ContactItem.PersonLastName
```

A loop statement references an entity and/or entity/attribute combination:

```
FOR EACH mPerson.Address
```

```
FOR EACH mPerson.FamilyRole WHERE mPerson.FamilyRole.LivesWith = "Y"
```

Position can be changed within an object instance:

```
SET CURSOR FIRST mPerson.FamilyRole WHERE mPerson.FamilyRole.LivesWith = "Y"
```

All the data for an object instance can be accessed with a single read statement, which not only generates all the SQL necessary to access the data, but sets up a work-in-progress object area in memory to hold the data while it is being processed:

```
ACTIVATE mPerson WHERE mPerson.ID = IContactList.Person.ID
```

Also, a single statement is all that is necessary to update the database with all the modified, added and deleted data that has been accumulated in a work-in-progress object.

```
COMMIT mPerson
```

Again, although the structure of those statements is procedural, they have all the power of being defined against the information tree structure of the IC Object, producing dramatic improvements in productivity and in the understanding of complex business rules.

## X. Human Interface Mapping

One of the major development tasks in business applications is the design and construction of the human interfaces that will present information to a user and allow that user to interact with the system. The information tree structure and extensions of ICA makes that a design-level visual experience and automates most of the detail required to build those interfaces that would normally be required using traditional programming techniques. The following shows how ICA addresses three different types of human interfaces: interactive Dialogs, Merge Documents/Emails and Graphs.

### Dialogs

Dialogs are groups of web pages or other interactive human interfaces to present and capture information from a user and guide that user through the online experience. In ICA, each page of a dialog is specified visually in that each individual control is “painted” on a frame representing the interactive page of the generated application. Each control references part of an IC Object by its object identifier (ie., its Object View Name) and its Entity Name and/or Attribute Name. Domain information tied to an attribute is used for formatting the data in the control.

Sample Table Domain Combo Box Control: In the following example, a combo box specifying “Citizenship” is placed on a group within a page (first image) and mapped to a single attribute, “mPerson.Demographics.CitizenshipCode” (second image). The table domain associated with the attribute formats the drop-down list during execution (third image).

The first screenshot shows a "Profile" form with various input fields. The "Citizenship" field is a dropdown menu. The second screenshot shows the "Combo Box Control Type - Domain" configuration window. The "Tag" is "CBCitizenship6", "Type" is "Domain", "Style" is "Drop Down List", and "Attribute Mapping" is set to "View: mPerson", "Entity: Demographics", "Attribute: CitizenshipCode". The third screenshot shows the dropdown menu for "Citizenship" with the following options: "Natural Born Citizen", "Naturalized Citizen", "Nonresident Alien", and "Resident Alien".

Field	Value
Historical ID	
Alien Reg Number	
Visa Classification	
Title	
Birth Country	
Previous Name	
Ethnicity	
Citizenship	
Occupation	
Gender	
Marital Status	
Deceased Date	5/27/2022
Primary Language	
Nation Of Citizenship	
Date of Birth	5/27/2022
City of Birth	
Maiden Name	
Deceased	<input type="checkbox"/>

Combo Box Control Type - Domain

Tag: CBCitizenship6

Type: Domain

Style: Drop Down List

Text:

☐ Sort Presented List ☐ User Specified List

☐ No Null Entry In List

Attribute Mapping

View: mPerson

Entity: Demographics

Attribute: CitizenshipCode

Context:

Field	Value
Ethnicity	American Indian/Alaska Ntv (N/Centra
Citizenship	Natural Born Citizen
Occupation	
Gender	F
Marital Status	
Deceased Date	



Sample Includable Subobject Combo Box Control: In the following example, a combo box specifying “Billing Year” is placed on a group within a page (first image). The combo box detail (second image) identifies an “Automatic Include” function from object mYearLST to billing object mSAPProf (part of which is shown in fourth image). The result during execution is shown in the third image. The definition of the combo box and the functionality of the IC Objects is all that is necessary to generate the solution into a target physical environment.

Given Name:  Middle Name:  Family Name:   
 Preferred Name:  Campus ID:   
 Government ID:  Financial Hold Date: 5/27/2022  
 Financial Hold:  Home Phone:   
 Preferred Email:  Primary Campus:   
 Work Phone:  Residency:   
 SA Cleared:  Billing Year:

Combo Box Control Type - Select

Tag: ComboBox1

Type: Automatic Include

Style: Drop Down List

Text:

☐ Sort Presented List

☐ No Null Entry In List

List Box Entity Mapping

Entity:

Scope:

☐ Scope Object Instance

Edit Control Mapping

View: mSAPProf

Entity: CurrentDegreeEntryCollegeYear

Attribute: Year

Context:

List Box Mapping

View: mYearLST

Entity: CollegeYear

Attribute: Year

Context:

Financial Hold Date

Home Phone 647.839.3513

Primary Campus MAIN

Residency Distance

Billing Year 2018-2019

2022-2023

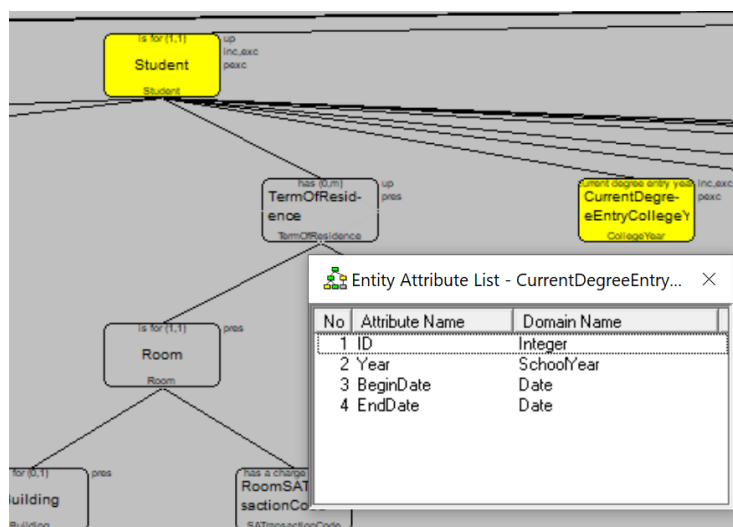
2021-2022

2020-2021

2019-2020


2018-2019

2017-2018




## Merge Documents/Emails

Merge Documents and Merge Emails are accomplished by creating a document template through an editor such as MS Word or ODT. The template can contain all of the formatting capability of the editor and is initially created with sample data. When that initial file is complete, the sample data is replaced with mapping attributes from an IC Object, including repeating data, similar to list boxes in dialogs and derived and work data within the object. A template example for a financial aid letter to a student is shown below in the first image. Note the specification for repeating AwardLetterFederalCOAItem entity data.

	<div>[Z:CollegeYear.Year] <b>Award Notification</b></div>
<hr/>	
[Z:Person.dToday]	IMU ID#: [Z:Person.CampusID]
 [Z:Person.dFullName] [Z:Address.dFullAddress]	
 <b>Cost of the [Z:CollegeYear.Year] Academic Year</b>	
Estimated Institutional Costs (\$[Z:AwardLetterFederal.CostOfAttendanceTotal])	
	<b>Fall                  Spring                  Full Year</b>
[Z:#S:AwardLetterFederalCOAItem]	[Z:AwardLetterFederalCOAItem.Description]
	[Z:AwardLetterFederalCOAItem.FallAmount][Z:AwardLetterFederalCOAItem.SpringAmount]
	[Z:AwardLetterFederalCOAItem.Amount]
[Z:#E]	
 <b>Grants and Scholarships</b>	
Total Grants and Scholarship(\$[Z:AwardLetterFederal.GrantsAndScholarshipsTotal])	
	<b>Fall                  Spring                  Full Year</b>
[Z:#S:AwardLetterFederalGrant]	[Z:AwardLetterFederalGrant.Description]
	[Z:AwardLetterFederalGrant.FallAmount][Z:AwardLetterFederalGrant.SpringAmount]
	[Z:AwardLetterFederalGrant.Amount]
[Z:#E]	
<hr/>	
Net Costs	(\$[Z:AwardLetterFederal.NetCosts])
Cost minus grants and scholarships	<b>Fall                  Spring                  Full Year</b>
	[Z:AwardLetterFederal.NetCostsFall][Z:AwardLetterFederal.NetCostsSpring]
	[Z:AwardLetterFederal.NetCosts]
<hr/>	
 <b>Options to Pay Net Costs</b>	
Work Options (\$[Z:AwardLetterFederal.WorkOptionsTotal])	
Federal/Texas Work-Study or regular campus student wages (must work to earn)	
(\$[Z:AwardLetterFederal.WorkOptionsTotal])	
	<b>Fall                  Spring                  Full Year</b>
	[Z:AwardLetterFederal.WorkOptionsFall][Z:AwardLetterFederal.WorkOptionsSpring]
	[Z:AwardLetterFederal.WorkOptionsTotal]
<hr/>	
Loan Options (\$[Z:AwardLetterFederal.LoanOptionsTotal])	
	<b>Fall                  Spring                  Full Year</b>
[Z:#S:AwardLetterFederalLoan]	[Z:AwardLetterFederalLoan.Description][Z:AwardLetterFederalLoan.FallAmount]
	[Z:AwardLetterFederalLoan.SpringAmount][Z:AwardLetterFederalLoan.Amount]
[Z:#E]	
<hr/>	
Final Net Costs	(\$[Z:AwardLetterFederal.FinalNetCosts])
Cost of attendance minus scholarships, grants, estimated work study and loans	<b>Fall                  Spring                  Full Year</b>
	[Z:AwardLetterFederal.FinalNetCostsFall][Z:AwardLetterFederal.FinalNetCostsSpring]
	[Z:AwardLetterFederal.FinalNetCosts]
<hr/>	
 <b>Other Options</b>	

*Imaginary U*  
125 years  
educating for  
success




*"For I know the  
thoughts that I think  
toward you, says the  
Lord, thoughts of  
peace and not of evil,  
to give you a future  
and a hope."*

*Jeremiah 29:11*

Knowledge. Faith. Service.

The next image shows a sample document generated from the template and the information within a financial aid IC Object instance. The information tree extensions provide a design-level capability that makes it straight-forward to use the power of the word processing function for generating finely formatted documents. The same capability can be used for generating emails.



IMAGINARY U

2021-2022  
**Award Notification**

---

August 13, 2021

IMU ID#: 269778

**William Martin**  
**9644 Nothing Road**  
**Houston , TX 77088**

### Cost of the 2021-2022 Academic Year

Estimated Institutional Costs (\$31,608.00)

	Fall	Spring	Full Year
Tuition	11,424.00	11,424.00	22,848.00
SA Fee	110.00	110.00	220.00
Technology Fee	220.00	220.00	440.00
Room and Board - On Campus	4,050.00	4,050.00	8,100.00

### Grants and Scholarships

Total Grants and Scholarship (\$7,000.00)

	Fall	Spring	Full Year
Achievement Award	1,000.00	1,000.00	2,000.00
Achievement Award+	500.00	500.00	1,000.00
Leadership Scholarship- Church	500.00	500.00	1,000.00
Imaginary U - Freshman Scholarship	1,500.00	1,500.00	3,000.00

### Net Costs

Cost minus grants and scholarships (\$24,608.00)

	Fall	Spring	Full Year
	12,304.00	12,304.00	24,608.00

### Options to Pay Net Costs

**Work Options**

Federal/Texas Work-Study or regular campus student wages (must work to earn)

	Fall	Spring	Full Year
	1,100.00	1,100.00	2,200.00

**Loan Options**

Federal Unsubsidized Direct Loan

	Fall	Spring	Full Year
	2,750.00	2,750.00	5,500.00

### Final Net Costs

Cost of attendance minus scholarships, grants, estimated work study and loans (\$16,908.00)


	Fall	Spring	Full Year
	8,454.00	8,454.00	16,908.00

### Other Options

Payment Plan offered by Imaginary U      Military and/or National Service benefits

Display Settings

**Imaginary U**  
125 years  
educating for  
success



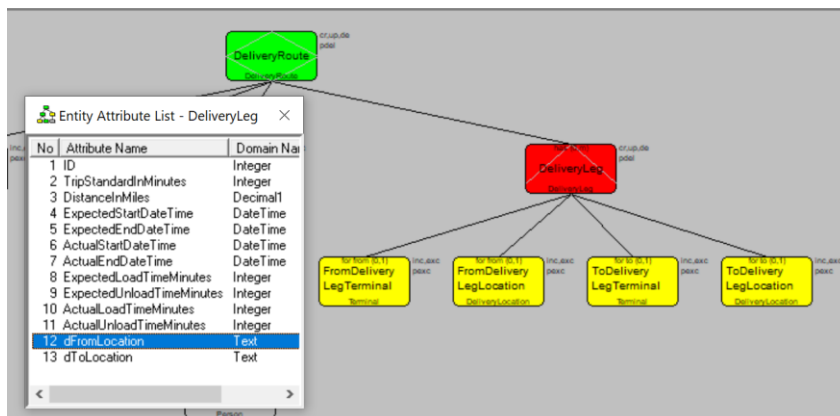
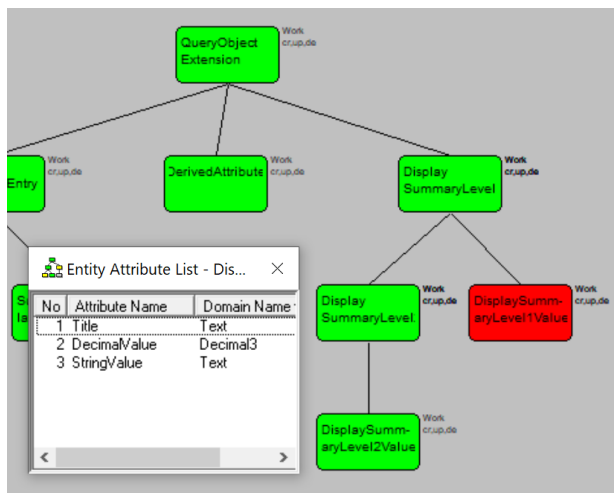
*"For I know the thoughts that I think toward you, says the Lord, thoughts of peace and not of evil, to give you a future and a hope."*

Jeremiah 29:11

Knowledge. Faith. Service.

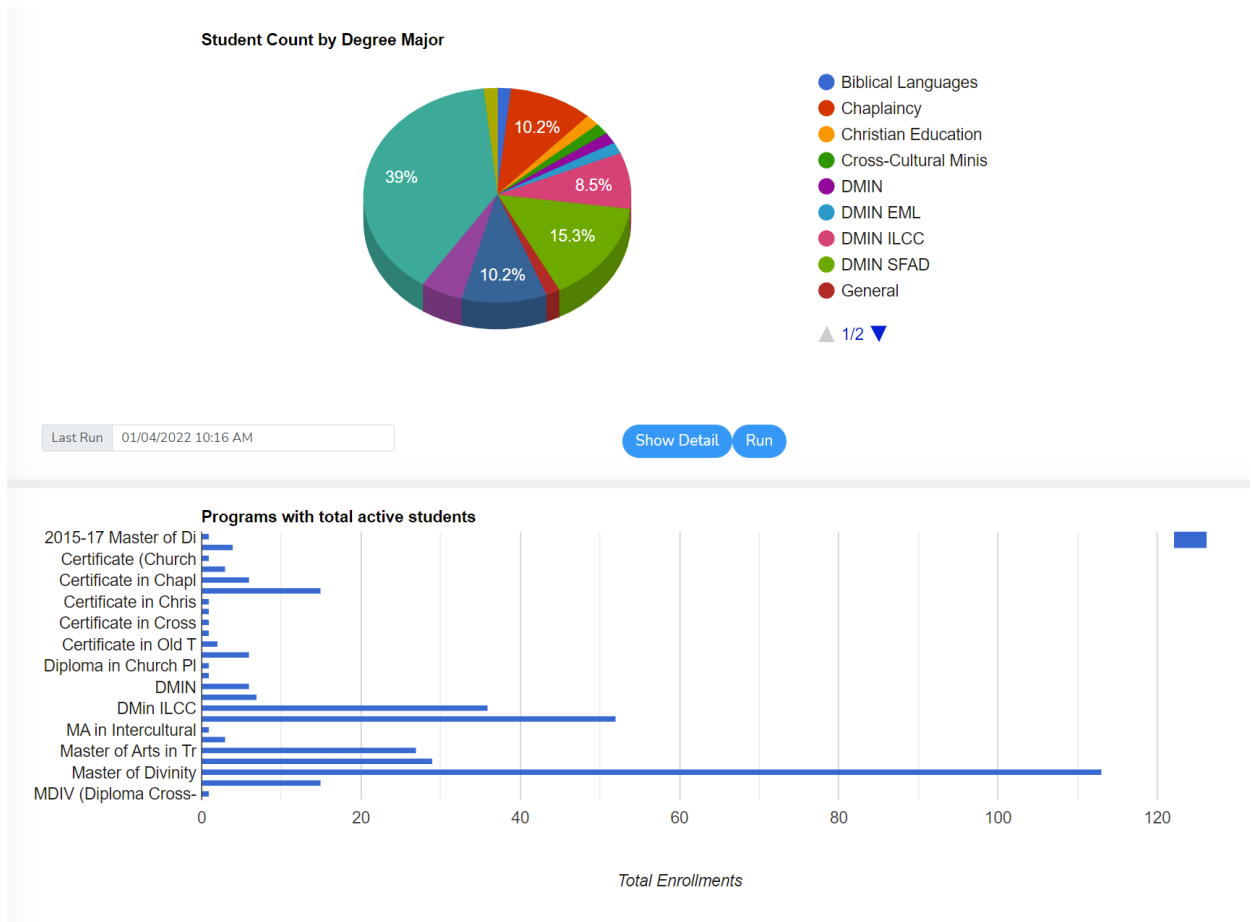
## Graphics

Graphic images are generated in ICA based upon the structure of a subobject within an IC Object. Different graphic images require different structures, but the most common images such as pie charts and single level bar charts just require a 2-level subobject (ie., a child under a parent). On the other hand, a 2-level bar chart requires a 3-level subobject. The first image below shows the IC Object for adding summary extensions to a Query Object and contains both a 2-level and 3-level subobject. The second image below shows part of a truck routing IC Object for formatting a Google map with routing information using common position data from the from/to Deliver Leg information.



Dashboards

When graphics images are combined with Query functionality, easy to use/change dashboards can be added to any ICA application, creating the kinds of images shown below.



## XI. Server-to-Server Communication

A growing requirement for business applications running in the cloud is the communication of information between cloud servers that need to share selective information. The ICA implementation uses information trees to enable this function, passing IC Objects across servers and using their integrity and processing capabilities to support and automate execution. Any IC Object can be sent from one server and processed by another as long as both servers have a copy of the IC Object definition. The structure and processing rules defined in the object enable execution. The integrity rules imbedded in the object definition manage integrity. For example, when a receiving server processes an IC Object and returns it to the sending server, there is nothing it can do that will cause the source data to be incorrectly updated because the sending server also has a definition of the integrity rules for the object and will only process the data accordingly.

**Samples of Communicating IC Object Information to Another Server:** The first sample below shows a web page for identifying proposed course offerings from remote schools that can be accepted for sharing by the host school. On striking the Send button, the user is taken to the second page where the IC Object for that information is edited and sent to the necessary other schools using a single SendObjectToServer operation.

Remote Courses Accepted by Host School						
200						
Send Save Add						
1 - 6 / 6						
< 1 >						
School	Course Number	Title	Credits	Accepted Date	Delete	
NTS	ACC101	Fundamentals of Accounting I	3.00	01/25/2022	✕	
NTS	ACC102	Fundamentals of Accounting II	3.00	01/25/2022	✕	
NTS	ACC105	Financial Accounting	3.00	01/25/2022	✕	
NTS	ACC200	Cost Accounting	3.00	01/25/2022	✕	
NTS	ACC301	International Accounting I	3.00	02/04/2022	✕	
NTS	ACC302	Environmental Accounting	3.00	02/04/2022	✕	
1 - 6 / 6						
< 1 >						

Academics > Administration > Accepted Courses Send		
Send to Selected Return		
50		
1 - 2 / 2		
< 1 >		
<input type="checkbox"/>	School	Address
<input checked="" type="checkbox"/>	ANU	http://192.168.1.106:8080/
<input type="checkbox"/>	LocalSchool	http://localhost/

## **XII. Merging Components**

The capability of Information-Centric Architecture that has the greatest impact on productivity is the merging of component libraries. This ability of ICA in turn depends on the merging capability of information trees of which four types are discussed below.

1. Merging two versions of the same IC Object – In this case, we have two different instances of the same object structure where the content of the source object is to merged into the target object. It is the option of the merge function to either keep or delete any content in the target that is not in the source. In all cases, new entity/attribute values in the source override those in the target. The information tree structure and enhancements make it very straightforward to merge complex data. Also, though not required, the entities in IC Objects are usually identified with a single generated ID attribute. This makes it easy to identify the same entity in two separate objects and allows all other attributes in an entity to be modifiable, eliminating the problems of composite foreign keys.
2. Merging subobject components of two different IC Objects – If two different objects have similar subobject structures, that subobject data can be merged from one object into another in the same manner as above.
3. Application source library support – One important example of merging is for the ICA tools themselves is the ability to the meta data of an ICA application. Though the ICA tools use an ER Data model to define meta components, current source management systems do not support keeping meta data in database form. Thus, the ICA tools keep the meta objects as separate files managed by a source management system and merge them as necessary, including during the run-time of the tools themselves.
4. Merging of components from one IC Library into another – This major capability of ICA is dependent on the fact that all meta data within an IC Library are themselves composed of IC Objects, as described in 3 above. The visual images of the information trees in both libraries of application data enable the logical understanding of the task itself and the use of information trees as meta data in the CIA tool set facilitate its execution.

## Conclusion

The Information Tree with its extensions that are implemented in IC Objects is the base for a consistent, high-level, information view of business data that supports the generation of business applications from design-level definitions. The results are:

1. Huge productivity gains from the ability to attack rigorous business problems at the design-level, generating executable systems and eliminating the low-level coding normally required by such problems.
2. Flexibility in generating solutions in multiple physical environments.
3. Flexibility in modifying and enhancing those business processes and interfaces.
4. Improved quality of application functionality as a result of keeping the focus of development on design-level issues and supporting iterative development.

However, in spite of the significant advantages of the information tree in building systems, the huge productivity gains promised by ICA will result from seeing the business application world through the lens of Reusable Component Libraries where applications are assembled from reusable components, all of this enabled through the information tree extensions of ICA.