Technical Reference

Inuendo 1.2 (Alpha release) – TECHNICAL REFERENCE

Technical Reference

Contents:

1. Preface

Inuendo, in short, is the virtualization of data. Traditionally, data has been imprisoned in physical tables with fixed record layouts. The limitations of such structure have inhibited developers and the users they support, for decades. By virtualizing the data instead, the majority of these limitations are eliminated, resulting in simpler development, maintenance and troubleshooting of applications.

Inuendo treats all business objects (a.k.a. Entities) the same, regardless of their role in the supply chain. This allows entities of lesser obvious importance to produce equally valuable business intelligence.

The blueprint of any Inuendo entity is a class definition comprised of properties – a concept similar to object oriented design. Each property conforms to a specified data type. When an entity is instantiated (created), its unique identity and high priority metadata are stored in a header record. The values of its properties are stored in an associative manner (by nickname) in a series of subtables – one table per data type. As the values of properties change over time, each change is recorded chronologically in the subtables, providing a time indexed history at the property level. This is extremely useful in troubleshooting and trend analysis. In addition, the scattering of data in this fashion provides a layer of security not possible with traditional linear record formats.

As its name and logo suggests, Inuendo begins and ends with I/O (input/output). Therefore, in order to provide applications a simple and consistent way to extract and manipulate its scattered data, it is accompanied by a wealth of open source API's to perform tasks such as entity creation and property value assignment. Various stored procedures produce result sets with useful collections of data organized from the header and subtables.

Despite the fact that Inuendo is being deployed in live applications, it is still a work in progress. Please consider joining our grass roots effort to virtualize data by becoming a member of our LinkedIn group and contributing your technical expertise to help us improve the functionality of Inuendo.

2. What's new?

Encrypted Date type (DATX)

Due to the sensitivity of Date of Birth and other dates, when it comes to identity protection, a new data type (DATX) has been added to the portfolio. It is accompanied by functions **getDatX**, **getDatXn**, **getDatXa** and **putDatX**. These values are actually stored as 40 character encrypted strings on disk.

Expansion of Number data type (NUMB)

In order to provide increased precision for latitudes and longitudes, the NUMB data type has been enlarged by two decimal positions, for a total length of 25, including 7 decimals.

Error Entities

All Inuendo PUT functions as well as the newEntity function will now capture information regarding failed output attempts in entities of new preloaded classes CLASSERROR and PROPERROR. This feature will likely expand in future releases.

- PUT functions will continue to imply a True value if they are successful. When unsuccessful, they will still imply a False value, but also create an associated PROPERROR entity containing information about the error.

- A successful newEntity function will continue to imply the new unique ID of the entity just created. When unsuccessful, it will imply a zero value and create a CLASSERROR entity containing information about the error.

- A new generic Error Report is available on the Inuendo main menu. The user is prompted for a Moment range and an optional User ID.

Partner Classes

One of the basic design principles of Inuendo is that all entities contain a Parent ID. In addition, some entities also have Link properties whose values represent the unique ID's of other entities. However, there has never been any governance regarding the class associated with these partner entities. This has been addressed by adding a Partner Class column to ENTPROP and adding a prompt for this value in Class Maintenance.

- When defining a Class header, you may now specify an optional Partner Class to indicate the required Class of the Parent ID when instantiating an Entity with the **newEntity** function.

- When defining a Link property, you may now specify an optional Partner Class to indicate the required Class of any EntityID assigned as the value argument of a **putLink** function on that property.

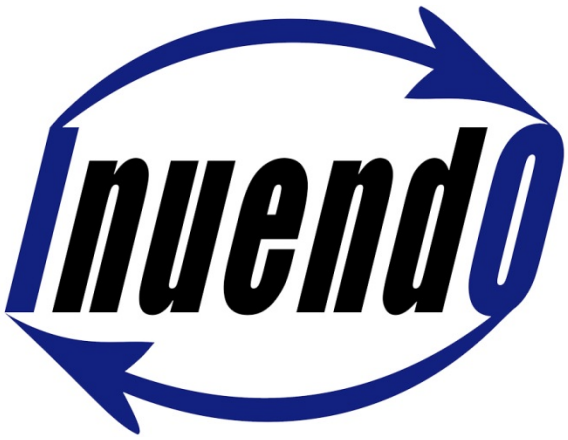
- If a Partner Class rule is violated in either case, the operation will fail and spawn an associated Error entity.

Entity Groups

A series of standardized stored procedures have been added to produce result sets of Entity Headers of a specified Class that are subordinate to a specified Parent entity, and which have a specified property value (either an exact match or within a range). This is intended to be a launching point for inquiry or reporting operations and eliminates the need to define SQL cursors for these tasks in application code. All time travel options are supported.

3. Data Types Legend

DATA TYPES LEGEND		
Inuendo type	**Native type**	**Primary uses**
entityID	Bigint	Unique ID or Parent ID in ENTHEAD. Entity ID in all data type subtables.
Datx (encrypted date)	Varchar(40)	VALUE column in subtable ENTDATX.
flag	Char(1)	VALUE column in subtable ENTFLAG. STATUS column in ENTHEAD.
job	Char(6)	CREATEJNUM column in ENTHEAD. CHANGEJNUM column in all data type subtables.
legacyN	Bigint	LEGACYN column in ENTHEAD.
link (same as entityID)	Bigint	VALUE column in subtable ENTLINK.
name	Char(10)	CREATEJNAM column in ENTHEAD. CHANGEJNAM column in all data type subtables.
nickname	Varchar(14)	CLASS column in ENTHEAD. PROPERTY column in all data type subtables.
note	Varchar(254)	VALUE column in subtable ENTNOTE. DESCRIPTOR column in ENTHEAD. Period delimited property path used in GET functions.
notx (encrypted note)	Varchar(510)	VALUE column in subtable ENTNOTX.
number	Decimal(25,7)	VALUE column in subtable ENTNUMB.
numx (encrypted number)	Varchar(62)	VALUE column in subtable ENTNUMX.
user	Char(18)	CHANGEUSER column in all data type subtables. CREATEUSER column in ENTHEAD.



4. Table structures

Table **ENTHEAD** – Entity Header

Contains one row for each Entity (instance of a Class). Once created, a row maybe updated but not deleted. The row is comprised of identification and metadata.

Column	Type	Description
EntityID	*entityID*	Database wide unique identifier (auto generated).
ParentID	*entityID*	The parent entity from which this entity was spawned and is subordinate to.
Class	*nickname*	The class of the entity as defined in ENTPROP.
LegacyA	*note*	The alpha legacy (well known) identifier of the entity.
LegacyN	*legacyN*	The numeric legacy (well known) identifier of the entity.
Descriptor	*note*	Freeform general description of the entity.
Status	*flag*	Simple flag for enabling/disabling entities or tracking their progression through a typical lifecycle.
CreateTime	*timestamp*	The system time stamp when the entity was created.
CreateJnam	*name*	IBM i job name which created the entity.
CreateUser	*user*	User ID which created the entity.
CreateJnum	*Job*	IBM i job number which created the entity.
CreateProg	*name*	IBM i program name which created the entity.
Primary key	EntityID	
Indexes	ENTHEADL1 (ParentID, Class, LegacyA, LegacyN) ENTHEADL2 (ParentID, Class, LegacyN) ENTHEADL3 (Class, LegacyA, LegacyN, EntityID) ENTHEADL4 (Class, LegacyN, EntityID)	

Table structures (cont.)

Table **ENTPROP** – Entity Properties

Contains one row for each Class (to provide a description), plus one row for each Property defined to each Class. GET and PUT functions use this table to verify Class and Property names prior to performing any I/O operations.

Column	Type	Description
Class	*nickname*	The name of the class (business object type).
Property	*nickname*	The name of the **Property**. Blank if simply describing the Class.
DataType	*nickname*	The data type for this **Property**. This will correspond with the LEGACYA value of a pre-installed entity of class DATATYPE.
Descriptor	*note*	Freeform general description of the **Property** (or **Class** if the **Property** is blank).
Sequencer	*smallint*	The value used to specify the order in which properties for this Class are presented by user interfaces.
PartnerClass	*nickname*	Optional. For a Class definition (Property is blank), specifies the required class of the Parent ID when instantiating an Entity of that Class. For a Property of type Link, specifies the required class for any value assigned to that Property.
Primary key	Class, Property	
Indexes	ENTPROPL1 (Class, Sequencer)	

Table structures (cont.)

Table **ENTDATE** – Entity Dates

Contains one for each historical change in value made to each Property of type DATE for each Class. The row contains time, user and program stamps for audit and time travel purposes.

Column	Type	Description
EntityID	*entityID*	Database wide unique identifier. Must exist in ENTHEAD.
Property	*nickname*	The associative name of the Property. Must be a valid Property nickname for the Class of this EntityID.
ChangeTime	*timestamp*	The system time stamp when the value was assigned.
ChangeJnam	*name*	IBM i job name which assigned this value to the property.
ChangeUser	*user*	User ID which assigned this value to the property.
ChangeJnum	*job*	IBM i job number which assigned this value to the property.
ChangeProg	*name*	IBM i program name which assigned this value to the property.
Value	*date*	The value assigned to the property for this entity.
Primary key	EntityID, Property, ChangeTime	

Table **ENTFLAG** – Entity Flags

Contains one for each historical change in value made to each Property of type FLAG for each Class. The row contains time, user and program stamps for audit and time travel purposes.

Column	Type	Description
EntityID	*entityID*	Database wide unique identifier. Must exist in ENTHEAD.
Property	*nickname*	The associative name of the Property. Must be a valid Property nickname for the Class of this EntityID.
ChangeTime	*timestamp*	The system time stamp when the value was assigned.
ChangeJnam	*name*	IBM i job name which assigned this value to the property.
ChangeUser	*user*	User ID which assigned this value to the property.
ChangeJnum	*job*	IBM i job number which assigned this value to the property.
ChangeProg	*name*	IBM i program name which assigned this value to the property.
Value	*flag*	The value assigned to the property for this entity.
Primary key	EntityID, Property, ChangeTime	

Table structures (cont.)

Table **ENTLINK** – Entity Links

Contains one for each historical change in value made to each Property of type LINK for each Class. The row contains time, user and program stamps for audit and time travel purposes.

Column	Type	Description
EntityID	*entityID*	Database wide unique identifier. Must exist in ENTHEAD.
Property	*nickname*	The associative name of the Property. Must be a valid Property nickname for the Class of this EntityID.
ChangeTime	*timestamp*	The system time stamp when the value was assigned.
ChangeJnam	*name*	IBM i job name which assigned this value to the property.
ChangeUser	*user*	User ID which assigned this value to the property.
ChangeJnum	*job*	IBM i job number which assigned this value to the property.
ChangeProg	*name*	IBM i program name which assigned this value to the property.
Value	*entityID*	The value assigned to the property for this entity. Must exist in ENTHEAD.
Primary key	EntityID, Property, ChangeTime	

Table **ENTNOTE** – Entity Notes

Contains one for each historical change in value made to each Property of type NOTE for each Class. The row contains time, user and program stamps for audit and time travel purposes.

Column	Type	Description
EntityID	*entityID*	Database wide unique identifier. Must exist in ENTHEAD.
Property	*nickname*	The associative name of the Property. Must be a valid Property nickname for the Class of this EntityID.
ChangeTime	*timestamp*	The system time stamp when the value was assigned.
ChangeJnam	*name*	IBM i job name which assigned this value to the property.
ChangeUser	*user*	User ID which assigned this value to the property.
ChangeJnum	*job*	IBM i job number which assigned this value to the property.
ChangeProg	*name*	IBM i program name which assigned this value to the property.
Value	*note*	The value assigned to the property for this entity.
Primary key	EntityID, Property, ChangeTime	

Table structures (cont.)

Table **ENTNOTX** – Entity Notes (encrypted)

Contains one for each historical change in value made to each Property of type NOTX for each Class. The row contains time, user and program stamps for audit and time travel purposes.

Column	Type	Description
EntityID	*entityID*	Database wide unique identifier. Must exist in ENTHEAD.
Property	*nickname*	The associative name of the Property. Must be a valid Property nickname for the Class of this EntityID.
ChangeTime	*timestamp*	The system time stamp when the value was assigned.
ChangeJnam	*name*	IBM i job name which assigned this value to the property.
ChangeUser	*user*	User ID which assigned this value to the property.
ChangeJnum	*job*	IBM i job number which assigned this value to the property.
ChangeProg	*name*	IBM i program name which assigned this value to the property.
Value	*notx*	The value assigned to the property for this entity, in encrypted form.
Primary key	EntityID, Property, ChangeTime	

Table **ENTNUMB** – Entity Numbers

Contains one for each historical change in value made to each Property of type NUMB for each Class. The row contains time, user and program stamps for audit and time travel purposes.

Column	Type	Description
EntityID	*entityID*	Database wide unique identifier. Must exist in ENTHEAD.
Property	*nickname*	The associative name of the Property. Must be a valid Property nickname for the Class of this EntityID.
ChangeTime	*timestamp*	The system time stamp when the value was assigned.
ChangeJnam	*name*	IBM i job name which assigned this value to the property.
ChangeUser	*user*	User ID which assigned this value to the property.
ChangeJnum	*job*	IBM i job number which assigned this value to the property.
ChangeProg	*name*	IBM i program name which assigned this value to the property.
Value	*number*	The value assigned to the property for this entity.
Primary key	EntityID, Property, ChangeTime	

Table **ENTNUMX** – Entity Numbers (encrypted)

Contains one for each historical change in value made to each Property of type NUMX for each Class. The row contains time, user and program stamps for audit and time travel purposes.

Encrypted numbers are stored on disk in a binary character format, but implied in a numeric form when retrieved from disk using the associated GET function.

Column	Type	Description
EntityID	*entityID*	Database wide unique identifier. Must exist in ENTHEAD.
Property	*nickname*	The associative name of the Property. Must be a valid Property nickname for the Class of this EntityID.
ChangeTime	*timestamp*	The system time stamp when the value was assigned.
ChangeJnam	*name*	IBM i job name which assigned this value to the property.
ChangeUser	*user*	User ID which assigned this value to the property.
ChangeJnum	*Job*	IBM i job number which assigned this value to the property.
ChangeProg	*Name*	IBM i program name which assigned this value to the property.
Value	*Numx*	The value assigned to the property for this entity, in encrypted form.
Primary key	EntityID, Property, ChangeTime	

Table **ENTDATX** – Entity Dates (encrypted)

Contains one for each historical change in value made to each Property of type DATX for each Class. The row contains time, user and program stamps for audit and time travel purposes.

Encrypted dates are stored on disk in a binary character format, but implied in a date form when retrieved from disk using the associated GET function.

Column	Type	Description
EntityID	*entityID*	Database wide unique identifier. Must exist in ENTHEAD.
Property	*nickname*	The associative name of the Property. Must be a valid Property nickname for the Class of this EntityID.
ChangeTime	*timestamp*	The system time stamp when the value was assigned.
ChangeJnam	*name*	IBM i job name which assigned this value to the property.
ChangeUser	*user*	User ID which assigned this value to the property.
ChangeJnum	*job*	IBM i job number which assigned this value to the property.
ChangeProg	*name*	IBM i program name which assigned this value to the property.
Value	*datx*	The value assigned to the property for this entity, in encrypted form.
Primary key	EntityID, Property, ChangeTime	

5. APPLICATION PROGRAM INTERFACES

- **Identity resolution functions**
- **Entity creation functions**
- **Subtable GET functions**
- **Subtable PUT functions**
- **Metadata GET functions**
- **Metadata PUT functions**
- **5250 Interface functions**
- **Miscellaneous functions**
- **Time Travel functions**
- **Statistical functions**
- **Audit trail procedures**
- **Entity Group procedures**

NOTE: Functions or procedures that accept an optional Moment argument will assume the current system time stamp if the argument is not specified. However, if a Session Moment is in effect, that value will be used instead of the current system time stamp when an optional Moment argument is not provided.

Identity resolution functions

These functions positively identify an entity based on its class, parent entity and legacy identifiers.

Function **getEntityID** – returns *entityID*	
getEntityID(**ParentID** *entityID*, **Class** *nickname*, **LegacyN** *bigint*, **LegacyA** *note*, **Moment** *timestamp*) getEntityID(**ParentID** *entityID*, **Class** *nickname*, **LegacyN** *bigint*, **LegacyA** *note*) getEntityID(**ParentID** *entityID*, **Class** *nickname*, **LegacyN** *bigint*, **Moment** *timestamp*) getEntityID(**ParentID** *entityID*, **Class** *nickname*, **LegacyN** *bigint*) getEntityID(**ParentID** *entityID*, **Class** *nickname*, **LegacyA** *note*, **Moment** *timestamp*) getEntityID(**ParentID** *entityID*, **Class** *nickname*, **LegacyA** *note*)	
Returns the unique ID of the entity of the specified **Class**, subordinate to the **ParentID** and having the specified numeric (**LegacyN**) or alpha legacy (**LegacyA**) identifiers, or a combination of both. The entity must have existed at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Under normal circumstances, an entity will have either a numeric or an alpha legacy ID, but not both.	
Exports for RPG	STDENTINP(GETENTITYID) STDENTINP(GETENTITYIDN) STDENTINP(GETENTITYIDA)
Exports for SQL	STDENTSQL1(GETENTITYID5) STDENTSQL1(GETENTITYID4) STDENTSQL1(GETENTITYIDN4) STDENTSQL1(GETENTITYIDN3) STDENTSQL1(GETENTITYIDA4) STDENTSQL1(GETENTITYIDA3)

Entity creation functions

These functions instantiate new entities for a specified Class, subordinate to a Parent entity, and optionally having a legacy identifier in either numeric or alpha format.

Function **newEntity** – returns *entityID*
newEntity(**ParentID** *entityID*, **Class** *nickname*, **LegacyN** *bigint*, **LegacyA** *note,* **Descriptor** *note*) newEntity(**ParentID** *entityID*, **Class** *nickname*, **LegacyN** *bigint*, **Descriptor** *note*) newEntity(**ParentID** *entityID*, **Class** *nickname*, **LegacyN** *bigint*) newEntity(**ParentID** *entityID*, **Class** *nickname*, **LegacyA** *note,* **Descriptor** *note*) newEntity(**ParentID** *entityID*, **Class** *nickname*, **LegacyA** *note*)
Creates a new entity of type **Class** in the ENTHEAD table. The new entity is subordinate to the **ParentID** and contains the legacy identifiers (**LegacyN**, **LegacyA**). Typically only one of the legacy identifiers contains a value. If a **Descriptor** is specified, it is applied to the ENTHEAD record as well. The new entity is also time, user and program stamped for audit and time travel purposes. In parallel, the arguments provided (metadata) are also written to designated data type subtables for audit and time travel purposes. Returns the unique ID of the new entity. See the **Metadata Put Functions** section for details on these parallel entries.

Exports for RPG	STDENTOUT(NEWENTITY) STDENTOUT(NEWENTITYN) STDENTOUT(NEWENTITYA)
Exports for SQL	STDENTSQL2(NEWENTITYB5) STDENTSQL2(NEWENTITYN4) STDENTSQL2(NEWENTITYN3) STDENTSQL2(NEWENTITYA4) STDENTSQL2(NEWENTITYA3)

Entity creation functions (cont.)

Function **dupEntity** – returns *entityID*	
dupEntity(**FromID** *entityID*, **LegacyN** *bigint*, **LegacyA** *note,* **Descriptor** *note*) dupEntity(**FromID** *entityID*, **LegacyN** *bigint*, **Descriptor** *note*) dupEntity(**FromID** *entityID*, **LegacyN** *bigint*) dupEntity(**FromID** *entityID*, **LegacyA** *note,* **Descriptor** *note*) dupEntity(**FromID** *entityID*, **LegacyA** *note*)	
Creates a new entity of the same class as **FromID**, with all properties set to an initial value matching the corresponding current values in **FromID**. Legacy identifiers **LegacyN** (numeric) and/or **LegacyA** (alpha) may be specified and will be placed in the corresponding columns of the new ENTHEAD record. An optional **Descriptor** may be provided for the new entity.	
Exports for RPG	STDENTOUT(DUPENTITY) STDENTOUT(DUPENTITYND) STDENTOUT(DUPENTITYN) STDENTOUT(DUPENTITYAD) STDENTOUT(DUPENTITYA)
Exports for SQL	STDENTOUT(DUPENTITY) STDENTOUT(DUPENTITYND) STDENTOUT(DUPENTITYN) STDENTOUT(DUPENTITYAD) STDENTOUT(DUPENTITYA)

Function **copyEntity** – returns *entityID*	
copyEntity(**FromID** *entityID*, **ParentID** *entityid*)	
Creates a new entity of the same class as **FromID**, subordinate to the specified **ParentID**, with all properties set to an initial value matching the corresponding current values in **FromID**. The remainder of the metadata (legacy identifiers, class and status) is copied verbatim to the new Entity header. Returns the unique ID of the new Entity.	
Exports for RPG	STDENTOUT(COPYENTITY)
Exports for SQL	STDENTOUT(COPYENTITY)

Subtable GET functions

These functions retrieve values from the data type subtables for the specified EntityID and Property nickname that was in effect at a specified Moment in time.

Function **getDate** – returns *date*	
getDate(**EntityID** *entityID*, **Property** *note*, **Moment** *timestamp*) getDate(**EntityID** *entityID*, **Property** *note*)	
Returns the VALUE column from subtable ENTDATE for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Property may be one of the following: • A valid nickname for a *date* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *date* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETDATE)
Exports for SQL	STDENTSQL1(GETDATE3) STDENTSQL1(GETDATE2)

Function **getDateN** – returns *numeric(8)*	
getDateN(**EntityID** *entityID*, **Property** *note*, **DateFormat** *char(5)*, **Moment** *timestamp*) getDateN(**EntityID** *entityID*, **Property** *note,* **DateFormat** *char(5)*)	
Returns the numeric cast of the VALUE column from subtable ENTDATE for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed. **DateFormat** is one of the valid IBM date formats, and represents the format of the return value.	
Property may be one of the following: • A valid nickname for a *date* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *date* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETDATEN)
Exports for SQL	STDENTSQL1(GETDATEN4) STDENTSQL1(GETDATEN3)

Subtable GET functions (cont.)

Function **getDateA** – returns *char(10)*	
getDateA(**EntityID** *entityID*, **Property** *note*, **DateFormat** *char(5)*, **Moment** *timestamp*) getDateA(**EntityID** *entityID*, **Property** *note,* **DateFormat** *char(5)*)	
Returns the character cast of the VALUE column from subtable ENTDATE for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed. **DateFormat** is one of the valid IBM date formats with an optional delimiter character, and represents the format of the return value.	
Property may be one of the following: • A valid nickname for a *date* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *date* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETDATEA)
Exports for SQL	STDENTSQL1(GETDATEA4) STDENTSQL1(GETDATEA3)

Function **getFlag** – returns *flag*	
getFlag(**EntityID** *entityID*, **Property** *note*, **Moment** *timestamp*) getFlag(**EntityID** *entityID*, **Property** *note*)	
Returns the VALUE column from subtable ENTFLAG for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Property may be one of the following: • A valid nickname for a *flag* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *flag* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETFLAG)
Exports for SQL	STDENTSQL1(GETFLAG3) STDENTSQL1(GETFLAG2)

Subtable GET functions (cont.)

Function **getLink** – returns *link*	
getLink(**EntityID** *entityID*, **Property** *note*, **Moment** *timestamp*) getLink(**EntityID** *entityID*, **Property** *note*)	
Returns the VALUE column from subtable ENTLINK for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Property may be one of the following: • A valid nickname for a *link* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *link* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETLINK)
Exports for SQL	STDENTSQL1(GETLINK3) STDENTSQL1(GETLINK2)

Function **getNote** – returns *note*	
getNote(**EntityID** *entityID*, **Property** *note*, **Moment** *timestamp*) getNote(**EntityID** *entityID*, **Property** *note*)	
Returns the VALUE column from subtable ENTNOTE for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Property may be one of the following: • A valid nickname for a *note* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *note* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETNOTE)
Exports for SQL	STDENTSQL1(GETNOTE3) STDENTSQL1(GETNOTE2)

Subtable GET functions (cont.)

Function **getNotX** – returns *note (w/decryption)*	
getNotX(**EntityID** *entityID*, **Property** *note*, **Moment** *timestamp*) getNotX(**EntityID** *entityID*, **Property** *note*)	
Returns the VALUE column from subtable ENTNOTX for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed. The value is automatically decrypted.	
Property may be one of the following: • A valid nickname for a *note* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *note* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETNOTE)
Exports for SQL	STDENTSQL1(GETNOTX3) STDENTSQL1(GETNOTX2)

Function **getNumb** – returns *number*	
getNumb(**EntityID** *entityID*, **Property** *note*, **Moment** *timestamp*) getNumb(**EntityID** *entityID*, **Property** *note*)	
Returns the VALUE column from subtable ENTNUMB for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Property may be one of the following: • A valid nickname for a *number* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *number* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETNUMB)
Exports for SQL	STDENTSQL1(GETNUMB) STDENTSQL1(GETNUMB)

Function **getNumX** – returns *number (w/decryption)*
getNumX(**EntityID** *entityID*, **Property** *note*, **Moment** *timestamp*) getNumX(**EntityID** *entityID*, **Property** *note*)
Returns the VALUE column from subtable ENTNUMX for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed. The value is automatically decrypted and converted to numeric format.
Property may be one of the following: • A valid nickname for a *note* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *note* type property associated with the class of the rightmost link property in the cascading path.

Exports for RPG	STDENTINP(GETNUMX)
Exports for SQL	STDENTSQL1(GETNUMX3) STDENTSQL1(GETNUMX2)

Function **getDatX** – returns *date (w/decryption)*
getDatX(**EntityID** *entityID*, **Property** *note*, **Moment** *timestamp*) getDatX(**EntityID** *entityID*, **Property** *note*)
Returns the VALUE column from subtable ENTDATX for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed. The value is automatically decrypted and converted to date format.
Property may be one of the following: • A valid nickname for a *note* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *note* type property associated with the class of the rightmost link property in the cascading path.

Exports for RPG	STDENTINP(GETDATX)
Exports for SQL	STDENTSQL1(GETDATX3) STDENTSQL1(GETDATX2)

Function **getDatxN** – returns *numeric(8)*	
getDatxN(**EntityID** *entityID*, **Property** *note*, **DateFormat** *char(5)*, **Moment** *timestamp*) getDatxN(**EntityID** *entityID*, **Property** *note,* **DateFormat** *char(5)*)	
Returns the numeric cast of the VALUE column from subtable ENTDATX for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed. **DateFormat** is one of the valid IBM date formats, and represents the format of the return value. The value is automatically decrypted and converted to numeric format.	
Property may be one of the following: • A valid nickname for a *date* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *date* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETDATXN)
Exports for SQL	STDENTSQL1(GETDATXN4) STDENTSQL1(GETDATXN3)

Function **getDatxA** – returns *char(10)*	
getDatxA(**EntityID** *entityID*, **Property** *note*, **DateFormat** *char(5)*, **Moment** *timestamp*) getDatxA(**EntityID** *entityID*, **Property** *note,* **DateFormat** *char(5)*)	
Returns the character cast of the VALUE column from subtable ENTDATX for the combination of **EntityID** and **Property** that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed. **DateFormat** is one of the valid IBM date formats with an optional delimiter character, and represents the format of the return value. The value is automatically decrypted and converted to character.	
Property may be one of the following: • A valid nickname for a *date* type property associated with the class of the **EntityID.** • A period delimited path of cascading link property nicknames followed by a valid nickname for a *date* type property associated with the class of the rightmost link property in the cascading path.	
Exports for RPG	STDENTINP(GETDATXA)
Exports for SQL	STDENTSQL1(GETDATXA4) STDENTSQL1(GETDATXA3)

Subtable PUT functions

These functions append the specified data type subtable with a new value entry for the specified EntityID and Property combination. The entry is time, user and program stamped for audit and time travel purposes.

Function **putDate** – returns *boolean*	
putDate(**EntityID** *entityID*, **Property** *note*, **NewValue** *date*)	
Appends the ENTDATE subtable with a time indexed entry for the specified **EntityID** and **Property**, reflecting the **NewValue**. Returns a true or false value indicating whether the operation was successful. If the **NewValue** matches the current value, no entry is written to ENTDATE, but the operation is still considered successful.	
Property must be a valid nickname for a *date* type property associated with the class of the **EntityID.**	
Exports for RPG	STDENTOUT(PUTDATE)
Exports for SQL	STDENTOUT(PUTDATE)

Function **putFlag** – returns *boolean*	
putFlag(**EntityID** *entityID*, **Property** *note*, **NewValue** *flag*)	
Appends the ENTFLAG subtable with a time indexed entry for the specified **EntityID** and **Property**, reflecting the **NewValue**. Returns a true or false value indicating whether the operation was successful. If the **NewValue** matches the current value, no entry is written to ENTFLAG, but the operation is still considered successful.	
Property must be a valid nickname for a *flag* type property associated with the class of the **EntityID.**	
Exports for RPG	STDENTOUT(PUTFLAG)
Exports for SQL	STDENTOUT(PUTFLAG)

Subtable PUT functions (cont.)

Function **putLink** – returns *boolean*	
putLink(**EntityID** *entityID*, **Property** *note*, **NewValue** *entityID*)	
Appends the ENTLINK subtable with a time indexed entry for the specified **EntityID** and **Property**, reflecting the **NewValue**. Returns a true or false value indicating whether the operation was successful. If the **NewValue** matches the current value, no entry is written to ENTLINK, but the operation is still considered successful.	
Property must be a valid nickname for a *link* type property associated with the class of the **EntityID.**	
Exports for RPG	STDENTOUT(PUTLINK)
Exports for SQL	STDENTOUT(PUTLINK)

Function **putNote** – returns *boolean*	
putNote(**EntityID** *entityID*, **Property** *note*, **NewValue** *note*)	
Appends the ENTNOTE subtable with a time indexed entry for the specified **EntityID** and **Property**, reflecting the **NewValue**. Returns a true or false value indicating whether the operation was successful. If the **NewValue** matches the current value, no entry is written to ENTNOTE, but the operation is still considered successful.	
Property must be a valid nickname for a *note* type property associated with the class of the **EntityID.**	
Exports for RPG	STDENTOUT(PUTNOTE)
Exports for SQL	STDENTOUT(PUTNOTE)

Subtable PUT functions (cont.)

Function **putNotX** – returns *boolean*	
putNotX(**EntityID** *entityID*, **Property** *note*, **NewValue** *note*)	
Appends the ENTNOTX subtable with a time indexed entry for the specified **EntityID** and **Property**, reflecting the **NewValue**. The value is automatically encrypted. Returns a true or false value indicating whether the operation was successful. If the **NewValue** matches the current value, no entry is written to ENTNOTX, but the operation is still considered successful.	
Property must be a valid nickname for a ***notx*** type property associated with the class of the **EntityID.**	
Exports for RPG	STDENTOUT(PUTNOTX)
Exports for SQL	STDENTOUT(PUTNOTX)

Function **putNumb** – returns *boolean*	
putNumb(**EntityID** *entityID*, **Property** *note*, **NewValue** *number*)	
Appends the ENTNUMB subtable with a time indexed entry for the specified **EntityID** and **Property**, reflecting the **NewValue**. Returns a true or false value indicating whether the operation was successful. If the **NewValue** matches the current value, no entry is written to ENTNUMB, but the operation is still considered successful.	
Property must be a valid nickname for a ***numb*** type property associated with the class of the **EntityID.**	
Exports for RPG	STDENTOUT(PUTNUMB)
Exports for SQL	STDENTOUT(PUTNUMB)

Function **putNumX** – returns *boolean*	
putNumX(**EntityID** *entityID*, **Property** *note*, **NewValue** *number*)	
Appends the ENTNUMX subtable with a time indexed entry for the specified **EntityID** and **Property**, reflecting the **NewValue**. The value is automatically encrypted. Returns a true or false value indicating whether the operation was successful. If the **NewValue** matches the current value, no entry is written to ENTNUMX, but the operation is still considered successful.	
Property must be a valid nickname for a ***numx*** type property associated with the class of the **EntityID.**	
Exports for RPG	STDENTOUT(PUTNUMX)
Exports for SQL	STDENTOUT(PUTNUMX)

Function **putDatX** – returns *boolean*	
putNumX(**EntityID** *entityID*, **Property** *note*, **NewValue** *date*)	
Appends the ENTDATX subtable with a time indexed entry for the specified **EntityID** and **Property**, reflecting the **NewValue**. The value is automatically encrypted. Returns a true or false value indicating whether the operation was successful. If the **NewValue** matches the current value, no entry is written to ENTDATX, but the operation is still considered successful.	
Property must be a valid nickname for a ***numx*** type property associated with the class of the **EntityID.**	
Exports for RPG	STDENTOUT(PUTDATX)
Exports for SQL	STDENTOUT(PUTDATX)

Metadata GET functions

These functions return current values from the ENTHEAD table for specified EntityID.

Function **getParentID** – returns *entityID*	
getParentID(**EntityID** *entityID)*	
Returns the **current** value of the PARENTID column from ENTHEAD for the specified **EntityID.** If a moment specific value is required, use the **getLink** function with property nickname **PARENTID** instead.	
Exports for RPG	STDENTINP(GETPARENTID)
Exports for SQL	STDENTSQL1(GETPARENTID)

Function **getClass** – returns *nickname*	
getClass(**EntityID** *entityID)*	
Returns the **current** value of the CLASS column from ENTHEAD for the specified **EntityID.** If a moment specific value is required, use the **getNote** function with property nickname **CLASS** instead.	
Exports for RPG	STDENTINP(GETCLASS)
Exports for SQL	STDENTSQL1(GETCLASS)

Function **getDescriptor** – returns *note*	
getDescriptor(**EntityID** *entityID)*	
Returns the **current** value of the DESCRIPTOR column from ENTHEAD for the specified **EntityID.** If a moment specific value is required, use the **getNote** function with property nickname **DESCRIPTOR** instead.	
Exports for RPG	STDENTINP(GETDESCRIPTOR)
Exports for SQL	STDENTSQL1(GETDESCRIPTOR)

Metadata GET functions (cont.)

Function **getLegacyA** – returns *note*	
getLegacyA(**EntityID** *entityID)*	
Returns the **current** value of the LEGACYA column from ENTHEAD for the specified **EntityID.** If a moment specific value is required, use the **getNote** function with property nickname **LEGACYA** instead.	
Exports for RPG	STDENTINP(GETLEGACYA)
Exports for SQL	STDENTSQL1(GETLEGACYA)

Function **getLegacyN** – returns *bigint*	
getLegacyN(**EntityID** *entityID)*	
Returns the **current** value of the LEGACYN column from ENTHEAD for the specified **EntityID.** If a moment specific value is required, use the **getNumb** function with property nickname **LEGACYN** instead.	
Exports for RPG	STDENTINP(GETLEGACYA)
Exports for SQL	STDENTSQL1(GETLEGACYA)

Function **getStatus** – returns *flag*	
getStatus(**EntityID** *entityID)*	
Returns the **current** value of the STATUS column from ENTHEAD for the specified **EntityID.** If a moment specific value is required, use the **getFlag** function with property nickname **STATUS** instead.	
Exports for RPG	STDENTINP(GETSTATUS)
Exports for SQL	STDENTSQL1(GETSTATUS)

Metadata PUT functions

These functions update the current values in the ENTHEAD table for specified EntityID. To provide an audit trial, each function also creates a parallel entry in a designated data type subtable.

Function **putParentID** – returns *boolean*	
putParentID(**EntityID** *entityID,* **NewParentID** *entityID)*	
Replaces the **current** value of the PARENTID column in ENTHEAD for the specified **EntityID.** In addition, unless this has been called by the RollBackJob function, an entry is written to the ENTLINK subtable using the property nickname PARENTID. Returns a true or false value indicating whether the operation was successful.	
Exports for RPG	STDENTOUT(PUTPARENTID)
Exports for SQL	STDENTOUT(PUTPARENTID)

Function **putClass** – returns *Boolean*	
putParentID(**EntityID** *entityID,* **NewClass** *nickname)*	
Replaces the **current** value of the CLASS column in ENTHEAD for the specified **EntityID.** In addition, unless this has been called by the RollBackJob function, an entry is written to the ENTNOTE subtable using the property nickname CLASS. Returns a true or false value indicating whether the operation was successful. Use with caution, because a change in CLASS could invalidate existing entries in the subtables if their property nicknames are not valid for the new CLASS.	
Exports for RPG	STDENTOUT(PUTCLASS)
Exports for SQL	STDENTOUT(PUTCLASS)

Metadata PUT functions (cont.)

Function **putDescriptor** – returns *Boolean*	
putDescriptor(**EntityID** *entityID,* **NewDescriptor** *note)*	
Replaces the **current** value of the DESCRIPTOR column in ENTHEAD for the specified **EntityID.** In addition, unless this has been called by the RollBackJob function, an entry is written to the ENTNOTE subtable using the property nickname DESCRIPTOR. Returns a true or false value indicating whether the operation was successful.	
Exports for RPG	STDENTOUT(PUTDESCRIPTOR)
Exports for SQL	STDENTOUT(PUTDESCRIPTOR)

Function **putLegacyN** – returns *Boolean*	
putLegacyN(**EntityID** *entityID,* **NewLegacyN** *bigint)*	
Replaces the **current** value of the LEGACYN column in ENTHEAD for the specified **EntityID.** In addition, unless this has been called by the RollBackJob function, an entry is written to the ENTNUMB subtable using the property nickname LEGACYN. Returns a true or false value indicating whether the operation was successful.	
Exports for RPG	STDENTOUT(PUTLEGACYN)
Exports for SQL	STDENTOUT(PUTLEGACYN)

Metadata PUT functions (cont.)

Function **putLegacyA** – returns *Boolean*	
putLegacyA(**EntityID** *entityID,* **NewLegacyA** *note)*	
Replaces the **current** value of the LEGACYA column in ENTHEAD for the specified **EntityID.** In addition, unless this has been called by the RollBackJob function, an entry is written to the ENTNOTE subtable using the property nickname LEGACYA. Returns a true or false value indicating whether the operation was successful.	
Exports for RPG	STDENTOUT(PUTLEGACYA)
Exports for SQL	STDENTOUT(PUTLEGACYA)

Function **putStatus** – returns *Boolean*	
putLegacyA(**EntityID** *entityID,* **NewStatus** *flag)*	
Replaces the **current** value of the STATUS column in ENTHEAD for the specified **EntityID.** In addition, unless this has been called by the RollBackJob function, an entry is written to the ENTFLAG subtable using the property nickname STATUS. Returns a true or false value indicating whether the operation was successful.	
Exports for RPG	STDENTOUT(PUTSTATUS)
Exports for SQL	STDENTOUT(PUTSTATUS)

5250 Interface functions

These functions provide convenience features for applications written for interactive 5250 access.

Function **PromptDate** – returns *date*	
PromptDate(**PromptText** *note,* **Required** *flag,* **DftValue** *date)* PromptDate(**PromptText** *note,* **Required** *flag)* PromptDate(**PromptText** *note)*	
Displays a pop-up window with the specified **PromptText**, requesting a date value. If **Required** is Y, a valid date value must be entered by the user. If **DftValue** is specified, the date field in the window is initialized to that value. Returns the date value.	
Exports for RPG	STD5250UTL(PROMPTDATE)

Function **PromptFlag** – returns *flag*	
PromptFlag(**PromptText** *note,* **Required** *flag,* **DftValue** *flag)* PromptFlag(**PromptText** *note,* **Required** *flag)* PromptFlag(**PromptText** *note)*	
Displays a pop-up window with the specified **PromptText**, requesting a flag value. If **Required** is Y, a flag value must be provided. If **DftValue** is specified, the flag field in the window is initialized to that value. Returns the flag value provided by the user.	
Exports for RPG	STD5250UTL(PROMPTFLAG)

5250 Interface functions (cont.)

Function **PromptNote** – returns *note*	
PromptNote(**PromptText** *note,* **Required** *flag,* **DftValue** *note)* PromptNote(**PromptText** *note,* **Required** *flag)* PromptNote(**PromptText** *note)*	
Displays a pop-up window with the specified **PromptText**, requesting a freeform note value in a four line array. If **Required** is Y, a note value must be entered by the user. If **DftValue** is specified, the note field in the window is initialized to that value. Returns the note value.	
Exports for RPG	STD5250UTL(PROMPTNOTE)

Function **PromptNumb** – returns *number*	
PromptNumb(**PromptText** *note,* **Required** *flag,* **DftValue** *number)* PromptNumb(**PromptText** *note,* **Required** *flag)* PromptNumb(**PromptText** *note)*	
Displays a pop-up window with the specified **PromptText**, requesting a number value. If **Required** is Y, a number value must be entered by the user. If **DftValue** is specified, the number field in the window is initialized to that value. Returns the number value.	
Exports for RPG	STD5250UTL(PROMPTNUMB)

Miscellaneous functions

These functions, while public, were created primarily for the purpose of enforcing integrity rules within the GET and PUT functions.

Function **ValidProperty** – returns *boolean*	
ValidProperty(**EntityID** *entityID,* **Property** *nickname,* **DataType** *nickname)* ValidProperty(**EntityID** *entityID,* **Property** *nickname)*	
Verifies that the specified **Property** nickname is valid for the class of the **EntityID**. If **DataType** is specified, the nickname must also be defined as that type. Returns a true or false value indicating whether the property nickname is valid.	
Exports for RPG	STDENTINP(VALIDPROPERTY)
Exports for SQL	STDENTSQL2(VALIDPROPERTY3) STDENTSQL2(VALIDPROPERTY2)

Function **ValidClassProperty** – returns *boolean*	
ValidClassProperty(**Class** *nickname,* **Property** *nickname,* **DataType** *nickname)* ValidClassProperty(**Class** *nickname,* **Property** *nickname)*	
Verifies that the specified **Property** nickname is valid for the specified **Class**. If **DataType** is specified, the nickname must also be defined as that data type. Returns a true or false value indicating whether the property nickname is valid.	
Exports for RPG	STDENTINP(VALIDCLASSPROPERTY)
Exports for SQL	STDENTSQL2(VALIDCLASSPROPERTY3) STDENTSQL2(VALIDCLASSPROPERTY2)

Function **PropertyOwner** – returns *entityID*	
PropertyOwner(**EntityID** *entityID,* **Property** *note)*	
Determines the unique ID of the entity actually used by GET functions to retrieve the value of the **Property** for the specified **EntityID**. This function takes into account any inheritance of properties from ancestor entities. The **Property** nickname may be prefixed by a period delimited path of link type property nicknames. Returns the actual entity ID of the ultimate property owner.	
Exports for RPG	STDENTINP(PROPERTYOWNER)
Exports for SQL	STDENTINP(PROPERTYOWNER)

Miscellaneous functions (cont.)

Function **Ancestor** – returns *entityID*	
Ancestor(**Descendant** *entityID,* **Class** *nickname)*	
Determines the unique ID of the entity of type **Class** that is a direct ancestor of the specified **Descendant**. The function travels upward in the ENTHEAD table using the ParentID at each level until it finds an entity with a matching **Class**. Returns the actual entity ID of that ancestor, or 0 if none is found.	
Exports for RPG	STDENTRULE(ANCESTOR)
Exports for SQL	STDENTRULE(ANCESTOR)

Function **HasChildren** – returns *Boolean*	
HasChildren(**EntityID** *entityID)*	
Determines whether any entities exist in ENTHEAD with a ParentID equal to the specified **EntityID**. Returns a true or false value indicating whether or not child entities were found. This function does NOT take into account matching values in the ENTLINK table.	
Exports for RPG	STDENTRULE(ANCESTOR)
Exports for SQL	STDENTRULE(ANCESTOR)

Function **ValidClass** – returns *boolean*	
ValidClass(**Class** *nickname)*	
Verifies that the specified **Class** nickname has been defined in the ENTPROP table. Returns a true or false value indicating whether the class nickname is valid.	
Exports for RPG	STDENTINP(VALIDCLASS)
Exports for SQL	None

Miscellaneous functions (cont.)

Function **FinalSegment** – returns *nickname*	
FinalSegment(**FullPath** *note)*	
Returns the final Property nickname in a period delimited property path. Used by the subtable GET functions to determine which Property value to retrieve from the actual Entity implied by the path.	
Exports for RPG	STDENTINP(FINALSEGMENT)
Exports for SQL	None

Procedure **SearchEntity**	
SearchEntity(**ParentID** *entityID,* **Class** *nickname,* **PreviousID** *entityID)* SearchEntity(**ParentID** *entityID,* **Class** *nickname)*	
For 5250 based applications, displays a pop-up window in which the user may select from a list of entities of the specified **Class** that are subordinate to the specified **ParentID**. The unique ID of the selected Entity is returned. If a **PreviousID** is specified and the user exits the window without making a selection, the **PreviousID** is returned.	
Exports for RPG	STDENTRULE(SEARCHENTITY)
Exports for SQL	None

Miscellaneous functions (cont.)

Function **DeleteEntity** – returns *boolean*	
DeleteEntity(**EntityID** *entityID*, **Cascading** *flag*) DeleteEntity(**EntityID** *entityID*)	
Deletes any subtable entries associated with the specified **EntityID**, then deletes as the header record in ENTHEAD. Also deletes any entries in the ENTLINK table whose value equals the **EntityID**. If dependent entities exist, the **Cascading** flag must be "Y", otherwise the function will terminate without any deletions and return a FALSE value. If **Cascading** = "Y", any dependent child entities, regardless of how many levels, are deleted as well. Returns a TRUE value if the header is successfully deleted. Note that referential constraints prohibit a header from being deleted if associated subtable entries or child entities exist. If **Cascading** is not specified, a value of "N" is assumed.	
Exports for RPG	STDENTUTIL(DELETEENTITY)
Exports for SQL	STDENTUTIL(DELETEENTITY2) STDENTUTIL(DELETEENTITY1)

Function **RollBackJob** – returns *boolean*	
RollBackJob(**JobName** *name*, **JobUser** *user,* **JobNumber** *job,* **Moment** *timestamp*) RollBackJob(**JobName** *name*, **JobUser** *user,* **JobNumber** *job*)	
For the specified IBM i job (**JobName, JobUser, JobNumber**), deletes the following: • Any subtable entries created by this job. If **Moment** is specified, only entries created since that **Moment** are deleted. If a subtable entry contains metadata, the ENTHEAD record for the associated entity is updated to reflect the prior value of that metadata. • Any entities created by this job. If **Moment** is specified, only entities created since that **Moment** are deleted. The end result is that the effects of the job since it began or since the specified **Moment** have been eliminated. A value of TRUE is returned if the operation is completely successful, otherwise a FALSE is returned. Note: No journaling or commitment control is required by this function.	
Exports for RPG	STDENTUTIL(ROLLBACKJOB)
Exports for SQL	STDENTUTIL(ROLLBACKJOB4) STDENTUTIL(ROLLBACKJOB3)

Miscellaneous functions (cont.)

Function **PrevMoment** – returns *timestamp*	
PrevMoment(**Moment** *timestamp*)	
Returns a time stamp that is one microsecond prior to the specified **Moment**. Intended for use by SQL functions and stored procedures due to the lack of microsecond level date arithmetic within SQL. Also used by the Temporal Integrity triggers when updating ENTHEAD metadata to a previous value.	
Exports for RPG	STDENTUTIL(PREVMOMENT)
Exports for SQL	STDENTUTIL(PREVMOMENT)

Function **ValidMethod** – returns *boolean*	
ValidMethod(**Method** *nickname)*	
Verifies that the specified **Method** nickname has been defined in the database SYSFUNCS table as a function. This function must accept an *EntityID* as its lone argument and imply one of the Inuendo data types as its result. See the section titled "Virtual Methods" for an explanation on how such functions may be invoked in an Inuendo expression. Returns a true or false value indicating whether the method nickname is valid.	
Exports for RPG	STDENTINP(VALIDMETHOD)
Exports for SQL	None

Function **UpperCase** – returns *note*	
UpperCase(**FreeText** *note*)	
Returns a simple upper case translation of the specified **FreeText**. Used primarily by the GET functions when validating *Class*, *Property* and *Method* names.	
Exports for RPG	STDENTINP(UPPERCASE)
Exports for SQL	None

Time Travel functions

Function **setSessionMoment** – returns *boolean*	
setSessionMoment(**SessionDate** *char(7),* **SessionTime** *char(6),* **SessionMicr** *integer*)	
Calculates a Moment in time using the specified **SessionDate** (CYYMMDD), **SessionTime** (HHMMSS) and **SessionMicr** (microseconds 0-999999), then places that time stamp inside the user space QTEMP/SETSSNMOM. The presence of this user space automatically disables all native Inuendo output operations. Used by the SETSSNMOM command. Returns a *true* value if the user space has been successfully set to the Moment value, *false* otherwise.	
Exports for RPG	STDENTINP(SETSESSIONMOMENT)
Exports for SQL	None

Function **getSessionMoment** – returns *timestamp*	
getSessionMoment()	
Returns the Moment value stored inside the user space QTEMP/SETSSNMOM, or a low value if the user space does not exist. Used by the DSPSSNMOM command.	
Exports for RPG	STDENTINP(GETSESSIONMOMENT)
Exports for SQL	None

Function **clrSessionMoment** – returns *boolean*	
clrSessionMoment()	
Deletes the user space QTEMP/SETSSNMOM. The absence of this user space automatically enables all native Inuendo output operations. Used by the CLRSSNMOM command. Returns a *true* value if the operation was successful, *false* otherwise.	
Exports for RPG	STDENTINP(CLRSESSIONMOMENT)
Exports for SQL	None

Time Travel functions (cont.)

Function **SessionMomentActive** – returns *boolean*	
SessionMomentActive()	
Returns a *true* value if the user space QTEMP/SETSSNMOM exists and contains a valid Moment value, *false* otherwise. Used by all native Inuendo output operations to determine whether they are disabled.	
Exports for RPG	STDENTINP(SESSIONMOMENTACTIVE)
Exports for SQL	None

Statistical functions

These functions perform basic aggregation on numeric property values at either the Entity or Parent level.

Property Average function series – returns *number*	
PropertyAvgDuring(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*, **ToMoment** *timestamp*) PropertyAvgThrough(**EntityID** *entityID*, **Property** *note*, **ToMoment** *timestamp*) PropertyAvgSince(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*) PropertyAvg(**EntityID** *entityID*, **Property** *note*)	
Returns the weighted average of each value held by the specified **Property** for the specified **EntityID**, based on the number of milliseconds each value was in effect over four distinct time periods: • **During**: The absolute starting **FromMoment** and ending **ToMoment** are specified. • **Through**: The **FromMoment** defaults to the Moment at which a value was first assigned to this **Property** and only the **ToMoment** is specified. • **Since**: The **ToMoment** defaults to the current system time and only the **FromMoment** is specified. • **None**: The **FromMoment** defaults to the Moment at which a value was first assigned to this **Property** and The **ToMoment** defaults to the current system time. Neither are specified. NOTE: Other aggregation types in this series will use the same four time periods as arguments.	
Exports for RPG	STDPRPAGGR(PROPERTYAVGDURING) STDPRPAGGR(PROPERTYAVGTHROUGH) STDPRPAGGR(PROPERTYAVGSINCE) STDPRPAGGR(PROPERTYAVG)
Exports for SQL	STDPRPAGGR(PROPERTYAVGDURING) STDPRPAGGR(PROPERTYAVGTHROUGH) STDPRPAGGR(PROPERTYAVGSINCE) STDPRPAGGR(PROPERTYAVG)

Property Minimum function series – returns *number*	
PropertyMinDuring(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*, **ToMoment** *timestamp*) PropertyMinThrough(**EntityID** *entityID*, **Property** *note*, **ToMoment** *timestamp*) PropertyMinSince(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*) PropertyMin(**EntityID** *entityID*, **Property** *note*)	
Returns the minimum value held by the specified **Property** for the specified **EntityID** at any time during the specified time period. The four intervals are the same as those used by the Property Average series:	
Exports for RPG	STDPRPAGGR(PROPERTYMINDURING) STDPRPAGGR(PROPERTYMINTHROUGH) STDPRPAGGR(PROPERTYMINSINCE) STDPRPAGGR(PROPERTYMIN)
Exports for SQL	STDPRPAGGR(PROPERTYMINDURING) STDPRPAGGR(PROPERTYMINTHROUGH) STDPRPAGGR(PROPERTYMINSINCE) STDPRPAGGR(PROPERTYMIN)

Property Maximum function series – returns *number*	
PropertyMaxDuring(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*, **ToMoment** *timestamp*) PropertyMaxThrough(**EntityID** *entityID*, **Property** *note*, **ToMoment** *timestamp*) PropertyMaxSince(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*) PropertyMax(**EntityID** *entityID*, **Property** *note*)	
Returns the maximum value held by the specified **Property** for the specified **EntityID** at any time during the specified time period. The four intervals are the same as those used by the Property Average series:	
Exports for RPG	STDPRPAGGR(PROPERTYMAXDURING) STDPRPAGGR(PROPERTYMAXTHROUGH) STDPRPAGGR(PROPERTYMAXSINCE) STDPRPAGGR(PROPERTYMAX)
Exports for SQL	STDPRPAGGR(PROPERTYMAXDURING) STDPRPAGGR(PROPERTYMAXTHROUGH) STDPRPAGGR(PROPERTYMAXSINCE) STDPRPAGGR(PROPERTYMAX)

Property Sum function series – returns *number*	
PropertySumDuring(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*, **ToMoment** *timestamp*) PropertySumThrough(**EntityID** *entityID*, **Property** *note*, **ToMoment** *timestamp*) PropertySumSince(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*) PropertySum(**EntityID** *entityID*, **Property** *note*)	
Returns the sum of all values assigned to the specified **Property** for the specified **EntityID** at any time during the specified time period. Used by the Mean series of functions, otherwise has little statistical relevance. The four intervals are the same as those used by the Property Average series:	
Exports for RPG	STDPRPAGGR(PROPERTYSUMDURING) STDPRPAGGR(PROPERTYSUMTHROUGH) STDPRPAGGR(PROPERTYSUMSINCE) STDPRPAGGR(PROPERTYSUM)
Exports for SQL	STDPRPAGGR(PROPERTYSUMDURING) STDPRPAGGR(PROPERTYSUMTHROUGH) STDPRPAGGR(PROPERTYSUMSINCE) STDPRPAGGR(PROPERTYSUM)

Property Count function series – returns *number*	
PropertyCountDuring(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*, **ToMoment** *timestamp*) PropertyCountThrough(**EntityID** *entityID*, **Property** *note*, **ToMoment** *timestamp*) PropertyCountSince(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*) PropertyCount(**EntityID** *entityID*, **Property** *note*)	
Returns the count of value assignments (PUT operations) of the specified **Property** for the specified **EntityID** at any time during the specified time period. Used by the Mean series of functions, otherwise has little statistical relevance, although the higher the count, the more volatile the property has been. The four intervals are the same as those used by the Property Average series:	
Exports for RPG	STDPRPAGGR(PROPERTYCOUNTDURING) STDPRPAGGR(PROPERTYCOUNTTHROUGH) STDPRPAGGR(PROPERTYCOUNTSINCE) STDPRPAGGR(PROPERTYCOUNT)
Exports for SQL	STDPRPAGGR(PROPERTYCOUNTDURING) STDPRPAGGR(PROPERTYCOUNTTHROUGH) STDPRPAGGR(PROPERTYCOUNTSINCE) STDPRPAGGR(PROPERTYCOUNT)

Property Mean function series – returns *number*	
PropertyMeanDuring(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*, **ToMoment** *timestamp*) PropertyMeanThrough(**EntityID** *entityID*, **Property** *note*, **ToMoment** *timestamp*) PropertyMeanSince(**EntityID** *entityID*, **Property** *note*, **FromMoment** *timestamp*) PropertyMean(**EntityID** *entityID*, **Property** *note*)	
Returns the non-weighted average of each value held by the specified **Property** for the specified **EntityID** within the specified time period, regardless of how long each value was in effect. The four intervals are the same as those used by the Property Average series:	
Exports for RPG	STDPRPAGGR(PROPERTYMEANDURING) STDPRPAGGR(PROPERTYMEANTHROUGH) STDPRPAGGR(PROPERTYMEANSINCE) STDPRPAGGR(PROPERTYMEAN)
Exports for SQL	STDPRPAGGR(PROPERTYMEANDURING) STDPRPAGGR(PROPERTYMEANTHROUGH) STDPRPAGGR(PROPERTYMEANSINCE) STDPRPAGGR(PROPERTYMEAN)

Entity Average – returns *number*	
EntityAvg(**EntityID** *entityID*, **Class** *nickname*, **Property** *note*, **Moment** *timestamp*) EntityAvg(**EntityID** *entityID*, **Class** *nickname*, **Property** *note*)	
Returns the straight average of all values of the specified **Property** for all entities of the specified **Class** that are subordinate to the specified **EntityID** (that is, their ParentID matches the specified **EntityID**), using the values in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Exports for RPG	STDENTAGGR(ENTITYAVGM) STDENTAGGR(ENTITYAVG)
Exports for SQL	STDENTAGGR(ENTITYAVGM) STDENTAGGR(ENTITYAVG)

Entity Minimum – returns *number*	
EntityMin(**EntityID** *entityID*, **Class** *nickname*, **Property** *note*, **Moment** *timestamp*) EntityMin(**EntityID** *entityID*, **Class** *nickname*, **Property** *note*)	
Returns the minimum of all values of the specified **Property** for all entities of the specified **Class** that are subordinate to the specified **EntityID** (that is, their ParentID matches the specified **EntityID**), using the values in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Exports for RPG	STDENTAGGR(ENTITYMINM) STDENTAGGR(ENTITYMIN)
Exports for SQL	STDENTAGGR(ENTITYMINM) STDENTAGGR(ENTITYMIN)

Entity Maximum – returns *number*	
EntityMax(**EntityID** *entityID*, **Class** *nickname*, **Property** *note*, **Moment** *timestamp*) EntityMax(**EntityID** *entityID*, **Class** *nickname*, **Property** *note*)	
Returns the maximum of all values of the specified **Property** for all entities of the specified **Class** that are subordinate to the specified **EntityID** (that is, their ParentID matches the specified **EntityID**), using the values in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Exports for RPG	STDENTAGGR(ENTITYMAXM) STDENTAGGR(ENTITYMAX)
Exports for SQL	STDENTAGGR(ENTITYMAXM) STDENTAGGR(ENTITYMAX)

Entity Sum – returns *number*	
EntitySum(**EntityID** *entityID*, **Class** *nickname*, **Property** *note*, **Moment** *timestamp*) EntitySum(**EntityID** *entityID*, **Class** *nickname*, **Property** *note*)	
Returns the sum of all values of the specified **Property** for all entities of the specified **Class** that are subordinate to the specified **EntityID** (that is, their ParentID matches the specified **EntityID**), using the values in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Exports for RPG	STDENTAGGR(ENTITYSUMM) STDENTAGGR(ENTITYSUM)
Exports for SQL	STDENTAGGR(ENTITYSUMM) STDENTAGGR(ENTITYSUM)

Entity Count – returns *number*	
EntitySum(**EntityID** *entityID*, **Class** *nickname*, **Moment** *timestamp*) EntitySum(**EntityID** *entityID*, **Class** *nickname*)	
Returns the count of all entities of the specified **Class** that are subordinate to the specified **EntityID** (that is, their ParentID matches the specified **EntityID**), that existed at the specified **Moment**. If **Moment** is not specified, the current system time stamp is assumed.	
Exports for RPG	STDENTAGGR(ENTITYCOUNTM) STDENTAGGR(ENTITYCOUNT)
Exports for SQL	STDENTAGGR(ENTITYCOUNTM) STDENTAGGR(ENTITYCOUNT)

Audit trail procedures

These procedures create result sets for use with client or web applications.

Procedure **Snapshot** – produces *result set* (**Property** *nickname*, **Descriptor** *note*, **Value** *note*)	
Snapshot(**EntityID** *entityID,* **Moment** *timestamp)* Snapshot(**EntityID** *entityID)*	
Produces a row for each **Property** defined in ENTPROP for the Class of the specified **EntityID**. Each row also contains the **Descriptor** associated with the **Property**, and the **Value** (cast as a character) that was in effect at the specified **Moment**. If **Moment** is not specified, the current system time stamp is used. The rows are presented in the order of the Sequencer field in ENTPROP.	
Exports for RPG	STDENTRULE(SNAPSHOT)
Exports for SQL	STDENTSQL1(SNAPSHOT2) STDENTSQL1(SNAPSHOT1)

Procedure **PropertyLife** – produces *result set* (**ChangeTime** *timestamp*, **ChangeUser** *user*, **ChangeJobName** *char(10)*, **ChangeJobNum** *char(6)*, **ChangeProgram** *char(10)*, **Value** *note*)	
PropertyLife(**EntityID** *entityID,* **Property** *nickname,* **Moment** *timestamp)* PropertyLife(**EntityID** *entityID,* **Property** *nickname)*	
Produces a row for each historical instance when the **Value** of the specified **Property** for the specified **EntityID** was reassigned. If **Moment** is specified, only the instances prior to that **Moment** are included. The rows contain the standard IBM i job identifiers (user, name and number) and are presented in timestamp sequence.	
Exports for RPG	STDENTRULE(PROPERTYLIFE)
Exports for SQL	STDENTSQL1(PROPERTYLIFE3) STDENTSQL1(PROPERTYLIFE2)

Audit trail procedures (cont.)

Procedure **EntityList** – produces *result set* (**Class** *nickname*, **EntityID** *entityID*, **Descriptor** *note*, **LegacyA** *note*, **LegacyN** *bigint*)
EntityList(**ParentID** *entityID,* **Class** *nickname,* **LegacyA** *note,* **LegacyN** *bigint,* **Descriptor** *note)*
Produces a row for each **EntityID** of the specified **Class** with a matching **ParentID**. Positioning arguments Descriptor, LegacyA and LegacyN are mutually exclusive. Specifying a value for one of these arguments will cause the list to be sorted on that column and contain only values equal to or greater than the argument. This procedure may be revisited in the future to allow for simpler invocations and Moment based selection. However, since **Descriptor** and **LegacyA** are the same type, overloading options are limited. Designed specifically for search utilities.

Exports for RPG	STDENTRULE(ENTITYLIST)
Exports for SQL	STDENTRULE(ENTITYLIST)

Procedure **EntityLife** – produces *result set* (**EntityID** *entityID*, **EventTime** *timestamp*, **EventUser** *name*, **EventJnam** *name*, **EventJnum** *job,* **EventProg** *name,* **Property** *nickname*, **DataType** *nickname*, **Value** *note*)
EntityLife(**EntityID** *entityID,* **Moment** *timestamp)*
Produces a row representing the initial creation of an **EntityID** plus one row for each historical assignment of any of its property values (which are expressed in this result set as a *note)*. If **Moment** is specified, only the instances prior to that **Moment** are included. The rows contain the standard IBM i job identifiers (user, name and number) and are presented in timestamp sequence.

Exports for RPG	STDENTRULE(ENTITYLIFE)
Exports for SQL	STDENTSQL2(ENTITYLIFE2) STDENTSQL2(ENTITYLIFE1)

Entity Group procedures

Procedure **EntityGroupDateRange** – produces *result set* (**ENTHEAD**)	
EntityGroupDateRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *date,* **RangeHigh** *date,* **Moment** *timestamp)* EntityGroupDateRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *date,* **RangeHigh** *date)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified date **Property** within the specified **Range** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to a date property of a linked Entity.	
Exports for RPG	STDENTRULE(ENTITYGROUPDATERANGEM) STDENTRULE(ENTITYGROUPDATERANGE) STDENTRULE(ENTITYGROUPDATE)
Exports for SQL	STDENTRULE(ENTITYGROUPDATERANGEM) STDENTRULE(ENTITYGROUPDATERANGE)

Procedure **EntityGroupDateValue** – produces *result set* (**ENTHEAD**)	
EntityGroupDateValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *date,* **Moment** *timestamp)* EntityGroupDateValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *date)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified date **Property** that matches the specified **Value** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to a date property of a linked Entity.	
Exports for RPG	STDENTRULE(ENTITYGROUPDATEVALUEM) STDENTRULE(ENTITYGROUPDATEVALUE) STDENTRULE(ENTITYGROUPDATE)
Exports for SQL	STDENTRULE(ENTITYGROUPDATEVALUEM) STDENTRULE(ENTITYGROUPDATEVALUE)

Entity Group procedures (cont.)

Procedure **EntityGroupFlagRange** – produces *result set* (**ENTHEAD**)	
EntityGroupFlagRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *flag,* **RangeHigh** *flag,* **Moment** *timestamp)* EntityGroupFlagRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *flag,* **RangeHigh** *flag)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified flag **Property** within the specified **Range** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to a flag property of a linked Entity.	
Exports for RPG	STDENTRULE(ENTITYGROUPFLAGRANGEM) STDENTRULE(ENTITYGROUPFLAGRANGE) STDENTRULE(ENTITYGROUPFLAG)
Exports for SQL	STDENTRULE(ENTITYGROUPFLAGRANGEM) STDENTRULE(ENTITYGROUPFLAGRANGE)

Procedure **EntityGroupFlagValue** – produces *result set* (**ENTHEAD**)	
EntityGroupFlagValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *flag,* **Moment** *timestamp)* EntityGroupFlagValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *flag)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified flag **Property** that matches the specified **Value** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to a flag property of a linked Entity.	
Exports for RPG	STDENTRULE(ENTITYGROUPFLAGVALUEM) STDENTRULE(ENTITYGROUPFLAGVALUE) STDENTRULE(ENTITYGROUPFLAG)
Exports for SQL	STDENTRULE(ENTITYGROUPFLAGVALUEM) STDENTRULE(ENTITYGROUPFLAGVALUE)

Entity Group procedures (cont.)

Procedure **EntityGroupNoteRange** – produces *result set* (**ENTHEAD**)	
EntityGroupNoteRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *note,* **RangeHigh** *note,* **Moment** *timestamp)* EntityGroupNoteRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *note,* **RangeHigh** *note)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified note **Property** within the specified **Range** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to a note property of a linked Entity.	
Exports for RPG	STDENTRULE(ENTITYGROUPNOTERANGEM) STDENTRULE(ENTITYGROUPNOTERANGE) STDENTRULE(ENTITYGROUPNOTE)
Exports for SQL	STDENTRULE(ENTITYGROUPNOTERANGEM) STDENTRULE(ENTITYGROUPNOTERANGE)

Procedure **EntityGroupNoteValue** – produces *result set* (**ENTHEAD**)	
EntityGroupNoteValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *note,* **Moment** *timestamp)* EntityGroupNoteValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *note)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified note **Property** that matches the specified **Value** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to a note property of a linked Entity.	
Exports for RPG	STDENTRULE(ENTITYGROUPNOTEVALUEM) STDENTRULE(ENTITYGROUPNOTEVALUE) STDENTRULE(ENTITYGROUPNOTE)
Exports for SQL	STDENTRULE(ENTITYGROUPNOTEVALUEM) STDENTRULE(ENTITYGROUPNOTEVALUE)

Entity Group procedures (cont.)

Procedure **EntityGroupNumbRange** – produces *result set* (**ENTHEAD**)	
EntityGroupNumbRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *number,* **RangeHigh** *number,* **Moment** *timestamp)* EntityGroupNumbRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *number,* **RangeHigh** *number)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified number **Property** within the specified **Range** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to a number property of a linked Entity.	
Exports for RPG	STDENTRULE(ENTITYGROUPNUMBRANGEM) STDENTRULE(ENTITYGROUPNUMBRANGE) STDENTRULE(ENTITYGROUPNUMB)
Exports for SQL	STDENTRULE(ENTITYGROUPNUMBRANGEM) STDENTRULE(ENTITYGROUPNUMBRANGE)

Procedure **EntityGroupNumbValue** – produces *result set* (**ENTHEAD**)	
EntityGroupNumbValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *number,* **Moment** *timestamp)* EntityGroupNumbValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *number)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified number **Property** that matches the specified **Value** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to a number property of a linked Entity.	
Exports for RPG	STDENTRULE(ENTITYGROUPNUMBVALUEM) STDENTRULE(ENTITYGROUPNUMBVALUE) STDENTRULE(ENTITYGROUPNUMB)
Exports for SQL	STDENTRULE(ENTITYGROUPNUMBVALUEM) STDENTRULE(ENTITYGROUPNUMBVALUE)

Entity Group procedures (cont.)

Procedure **EntityGroupNotxRange** – produces *result set* (**ENTHEAD**)	
EntityGroupNotxRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *note,* **RangeHigh** *note,* **Moment** *timestamp)* EntityGroupNotxRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *note,* **RangeHigh** *note)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified encrypted note **Property** within the specified **Range** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to an encrypted note property of a linked Entity. The **Range** arguments are passed in unencrypted form.	
Exports for RPG	STDENTRULE(ENTITYGROUPNOTXRANGEM) STDENTRULE(ENTITYGROUPNOTXRANGE) STDENTRULE(ENTITYGROUPNOTX)
Exports for SQL	STDENTRULE(ENTITYGROUPNOTXRANGEM) STDENTRULE(ENTITYGROUPNOTXRANGE)

Procedure **EntityGroupNotxValue** – produces *result set* (**ENTHEAD**)	
EntityGroupNotxValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *note,* **Moment** *timestamp)* EntityGroupNotxValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *note)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified encrypted note **Property** that matches the specified **Value** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to an encrypted note property of a linked Entity. The **Value** argument is passed in unencrypted form.	
Exports for RPG	STDENTRULE(ENTITYGROUPNOTXVALUEM) STDENTRULE(ENTITYGROUPNOTXVALUE) STDENTRULE(ENTITYGROUPNOTX)
Exports for SQL	STDENTRULE(ENTITYGROUPNOTXVALUEM) STDENTRULE(ENTITYGROUPNOTXVALUE)

Entity Group procedures (cont.)

Procedure **EntityGroupNumxRange** – produces *result set* (**ENTHEAD**)	
EntityGroupNumxRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *number,* **RangeHigh** *number,* **Moment** *timestamp)* EntityGroupNumxRange(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **RangeLow** *number,* **RangeHigh** *number)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified encrypted number **Property** within the specified **Range** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to an encrypted number property of a linked Entity. The **Range** arguments are passed in unencrypted form.	
Exports for RPG	STDENTRULE(ENTITYGROUPNUMXRANGEM) STDENTRULE(ENTITYGROUPNUMXRANGE) STDENTRULE(ENTITYGROUPNUMX)
Exports for SQL	STDENTRULE(ENTITYGROUPNUMXRANGEM) STDENTRULE(ENTITYGROUPNUMXRANGE)

Procedure **EntityGroupNumxValue** – produces *result set* (**ENTHEAD**)	
EntityGroupNumxValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *number,* **Moment** *timestamp)* EntityGroupNumxValue(**ParentID** *entityID,* **Class** *nickname,* **Property** *note,* **Value** *number)*	
Produces a row in the same format as table ENTHEAD for each **EntityID** of the specified **Class** subordinate to the specified **ParentID**, and having a value for the specified encrypted number **Property** that matches the specified **Value** in effect at the specified **Moment**. If no **Moment** is specified, the current system time stamp or Session Moment (if active) is used. The **Property** may be a period delimited path to an encrypted number property of a linked Entity. The **Value** argument is passed in unencrypted form.	
Exports for RPG	STDENTRULE(ENTITYGROUPNUMXVALUEM) STDENTRULE(ENTITYGROUPNUMXVALUE) STDENTRULE(ENTITYGROUPNUMX)
Exports for SQL	STDENTRULE(ENTITYGROUPNUMXVALUEM) STDENTRULE(ENTITYGROUPNUMXVALUE)

5. Virtual Methods

Virtual methods are a means to leverage new or existing business logic seamlessly through the Inuendo GET functions. It involves SQL user defined functions (UDF) in the Inuendo schema, which either perform the business logic themselves or link to executable units written in other high level languages, capable of implying a return value. Such UDF's are defined using the CREATE FUNCTION statement in SQL, or an equivalent graphical wizard, such as IBM i Navigator or IBM Data Studio.

Once these UDF's have been created, their names may be used by the GET functions as either the lone property name or the final segment in a period delimited path. When the GET functions attempt to validate the Property name argument, they first determine whether or not a UDF of the same name exists. If it does, it will execute that function, passing the EntityID (or the EntityID implied by the period delimited property path itself) as the lone argument. Therefore, these SQL functions support only a single argument of type *EntityID*.

The UDF must imply a return value compatible with one of the Inuendo data types. Likewise, only the Inuendo GET function associated with this data type should reference the UDF name in its period delimited property path. Otherwise the GET function will return a neutral value, similar to situations where an invalid property name is passed as an argument.

Example:

An IBM i service program contains an exported procedure LifetimeSales, which analyzes customer order history and sums of the monetary value of all shipments for either an individual ship-to location, or the sum total of all ship-to locations. It returns a decimal(15,2) value. The UDF definition links the SQL name "LifetimeSales" to the exported procedure, thereby making the name "LifeTimeSales" eligible to use as either a property (or the last segment of a period delimited property path) on a getNumb function call, because getNumb returns a compatible decimal(25,7) value. Assume that the ORDER class contains a LINK property called CUSTOMER).

```
AllTimeSales = getNumb(CustomerID, 'LifetimeSales');          // or : separator for RPG

AllTimeSales = getNumb(OrderID, 'CUSTOMER.LIFETIMESALES');    // or : separator for RPG
```

6. Temporal Integrity

In order to ensure that the true state of all Inuendo entities and their associated properties is accurate for any moment in time, it is necessary to prevent any I/O operations on Inuendo tables performed outside the scope of the PUT family of functions. In addition, when subtable entries representing metadata are deleted as a result of the RollBackJob function, the associated ENTHEAD record must be updated to reflect the previous value of that metadata.

A series of triggers have been established by the installation process to protect the integrity of the data, relative to the moment at which it was created or updated.

Trigger program **TmpIntHead** – For I/O events on ENTHEAD
For inserts: None.
For deletes: If the delete occurs because of the **DeleteEntity** function, the operation is allowed and no action is taken. Otherwise the ENTHEAD record is re-inserted record exactly as it was, including the original EntityID. Name: TMPINTHEADD
For updates: If the update occurs because of one of the metadata PUT functions (PutClass, PutParentID, PutDescriptor, PutLegacyN, PutLegacyA or PutStatus), the operation is allowed and no action is taken. Otherwise the ENTHEAD record retains its “before” image, thereby negating the update. Name: TMPINTHEADU

Temporal integrity (cont.)

Trigger program **TmpIntDate** – For I/O events on subtable ENTDATE
For inserts: None.
For deletes: If the delete occurs because of either the **DeleteEntity** or the **RollBackJob** function, the operation is allowed and no action is taken. Otherwise the ENTDATE record is re-inserted record exactly as it was. Name: TMPINTDATED
For updates: The ENTDATE record retains its "before" image, thereby negating the update. Name: TMPINTDATEU

Trigger program **TmpIntFlag** – For I/O events on subtable ENTFLAG
For inserts: None.
For deletes: If the delete occurs because of either the **DeleteEntity** or the **RollBackJob** function, the operation is allowed. Otherwise the ENTFLAG record is re-inserted record exactly as it was. When the delete is allowed and the property nickname represents metadata (**Status**), the associated metadata PUT function is used to update ENTHEAD with the most recent value for that metadata property. Name: TMPINTFLAGD
For updates: The ENTFLAG record retains its "before" image, thereby negating the update. Name: TMPINTFLAGU

Temporal integrity (cont.)

Trigger program **TmpIntLink** – For I/O events on subtable ENTLINK
For inserts: None.
For deletes: If the delete occurs because of either the **DeleteEntity** or the **RollBackJob** function, the operation is allowed. Otherwise the ENTLINK record is re-inserted record exactly as it was. When the delete is allowed and the property nickname represents metadata (**ParentID**), the associated metadata PUT function is used to update ENTHEAD with the most recent value for that metadata property. Name: TMPINTLINKD
For updates: The ENTLINK record retains its "before" image, thereby negating the update. Name: TMPINTLINKU

Trigger program **TmpIntNote** – For I/O events on subtable ENTNOTE
For inserts: None.
For deletes: If the delete occurs because of either the **DeleteEntity** or the **RollBackJob** function, the operation is allowed. Otherwise the ENTNOTE record is re-inserted record exactly as it was. When the delete is allowed and the property nickname represents metadata (**Class, Descriptor, LegacyA**), the associated metadata PUT function is used to update ENTHEAD with the most recent value for that metadata property. Name: TMPINTNOTED
For updates: The ENTNOTE record retains its "before" image, thereby negating the update. Name: TMPINTNOTEU

Temporal integrity (cont.)

Trigger program **TmpIntNotx** – For I/O events on subtable ENTNOTX
For inserts: None.
For deletes: If the delete occurs because of either the **DeleteEntity** or the **RollBackJob** function, the operation is allowed and no action is taken. Otherwise the ENTNOTX record is re-inserted record exactly as it was. Name: TMPINTNOTXD
For updates: The ENTNOTX record retains its "before" image, thereby negating the update. Name: TMPINTNOTXU

Trigger program **TmpIntNumb** – For I/O events on subtable ENTNUMB
For inserts: None.
For deletes: If the delete occurs because of either the **DeleteEntity** or the **RollBackJob** function, the operation is allowed. Otherwise the ENTNUMB record is re-inserted record exactly as it was. When the delete is allowed and the property nickname represents metadata (**LegacyN**), the associated metadata PUT function is used to update ENTHEAD with the most recent value for that metadata property. Name: TMPINTNUMBD
For updates: The ENTNUMB record retains its "before" image, thereby negating the update. Name: TMPINTNUMBU

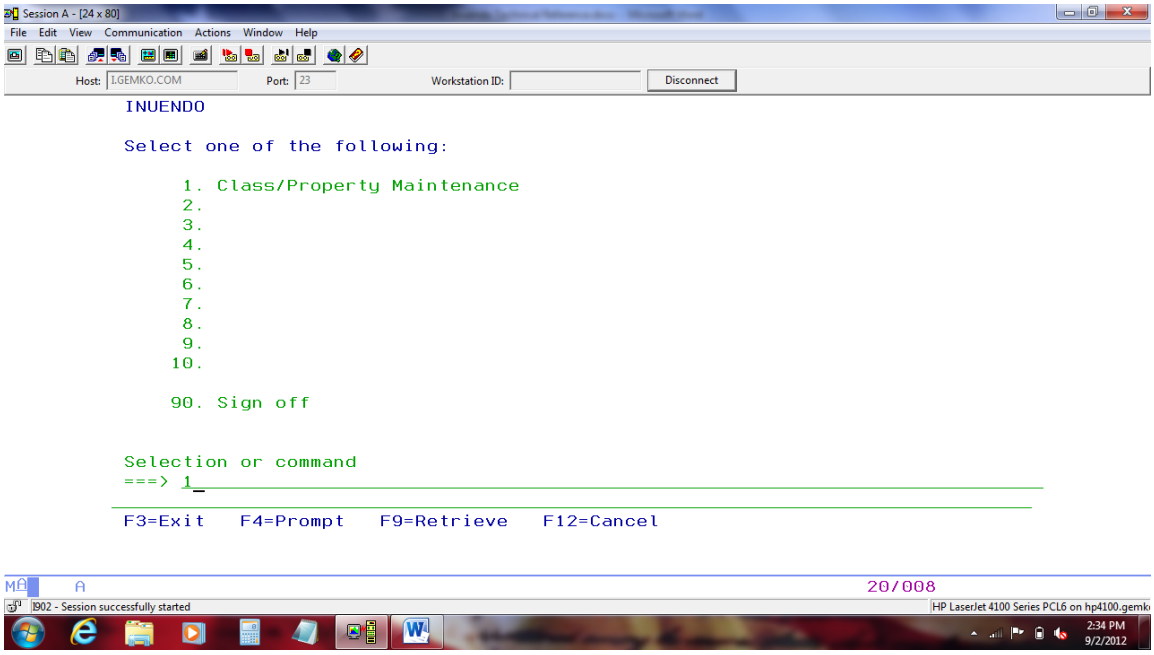
Temporal integrity (cont.)

Trigger program **TmpIntNumx** – For I/O events on subtable ENTNUMX
For inserts: None.
For deletes: If the delete occurs because of either the **DeleteEntity** or the **RollBackJob** function, the operation is allowed and no action is taken. Otherwise the ENTNUMX record is re-inserted record exactly as it was. Name: TMPINTNUMXD
For updates: The ENTNUMX record retains its “before” image, thereby negating the update. Name: TMPINTNUMXU

Trigger program **TmpIntDatx** – For I/O events on subtable ENTDATX
For inserts: None.
For deletes: If the delete occurs because of either the **DeleteEntity** or the **RollBackJob** function, the operation is allowed and no action is taken. Otherwise the ENTDATX record is re-inserted record exactly as it was. Name: TMPINTDATXD
For updates: The ENTNUMB record retains its “before” image, thereby negating the update. Name: TMPINTDATXU

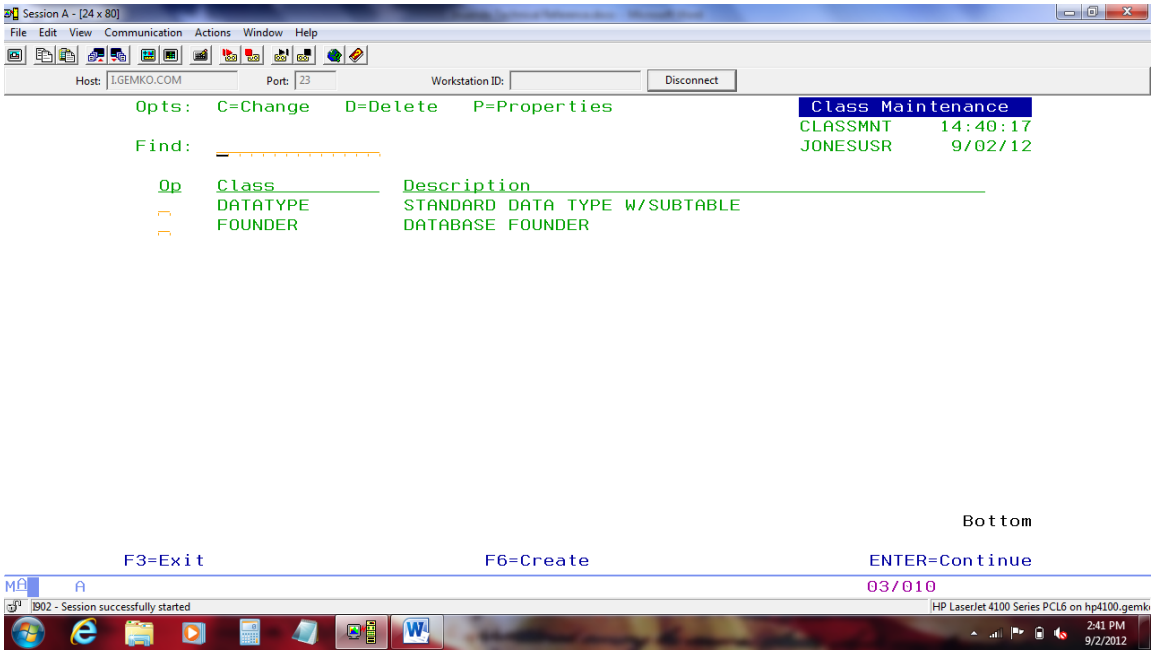
7. Class maintenance utility (5250 based)

Until a browser based equivalent is complete, this is the primary means to define Classes and their associated Properties (if required) in the ENTPROP table. It is accessible via Option 1 on the Inuendo main menu.

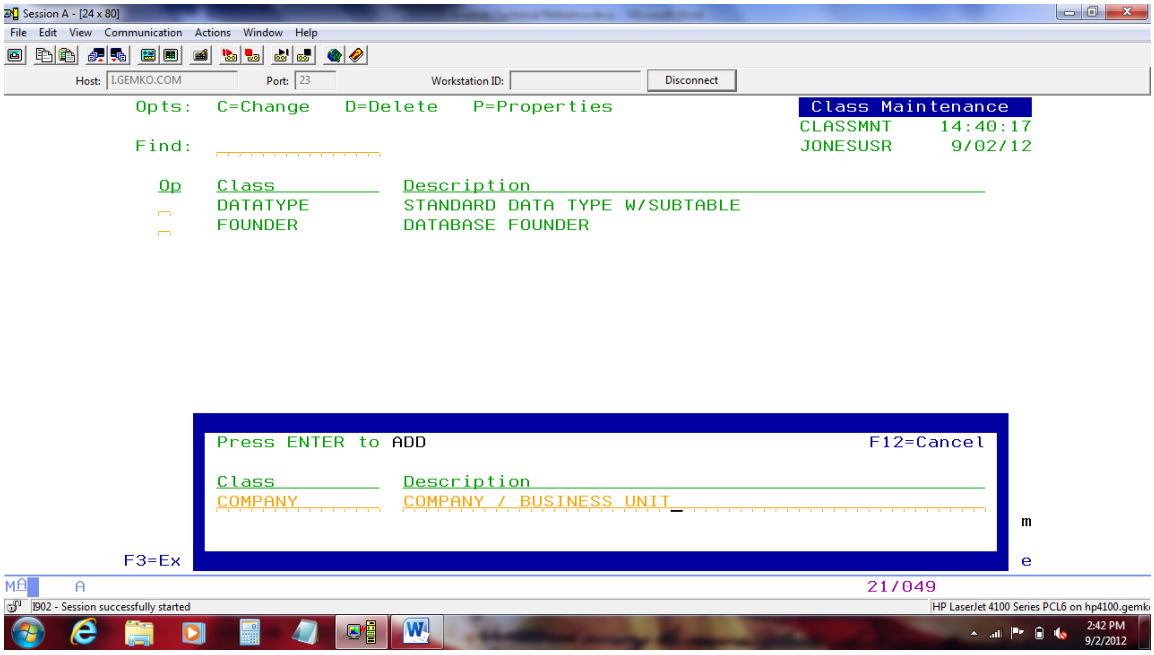


Class maintenance utility (cont.)

The list of defined Classes is displayed. Note that DATATYPE and FOUNDER are automatically loaded upon installation. To define a new Class, press F6:

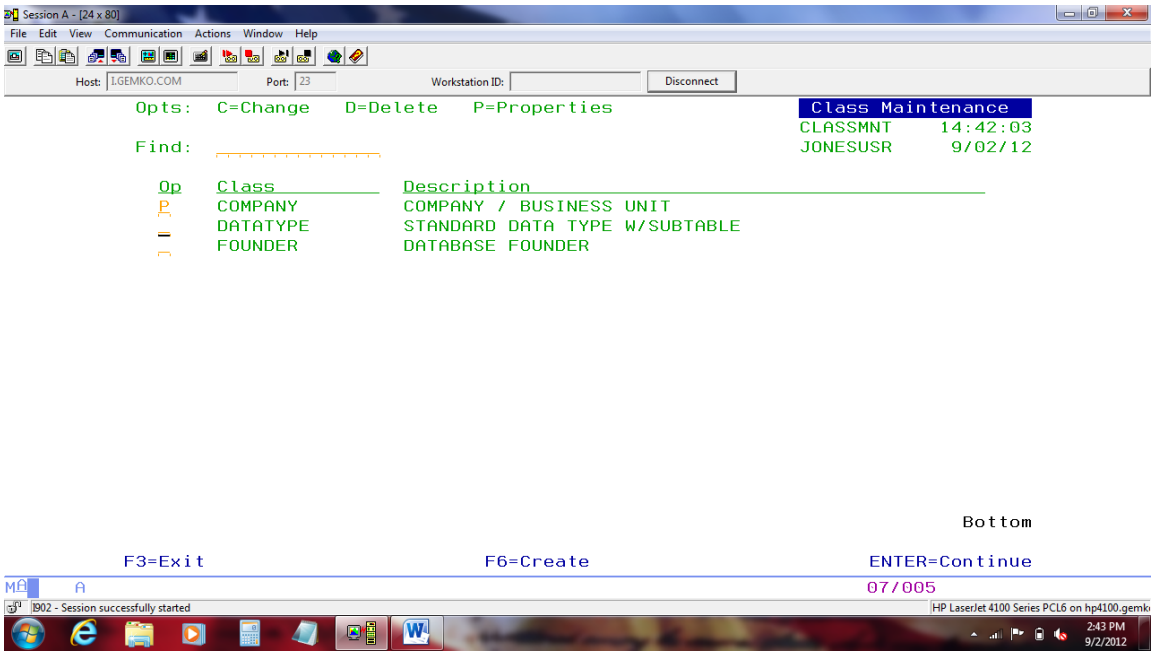


Fill in the Class name and description and press Enter:

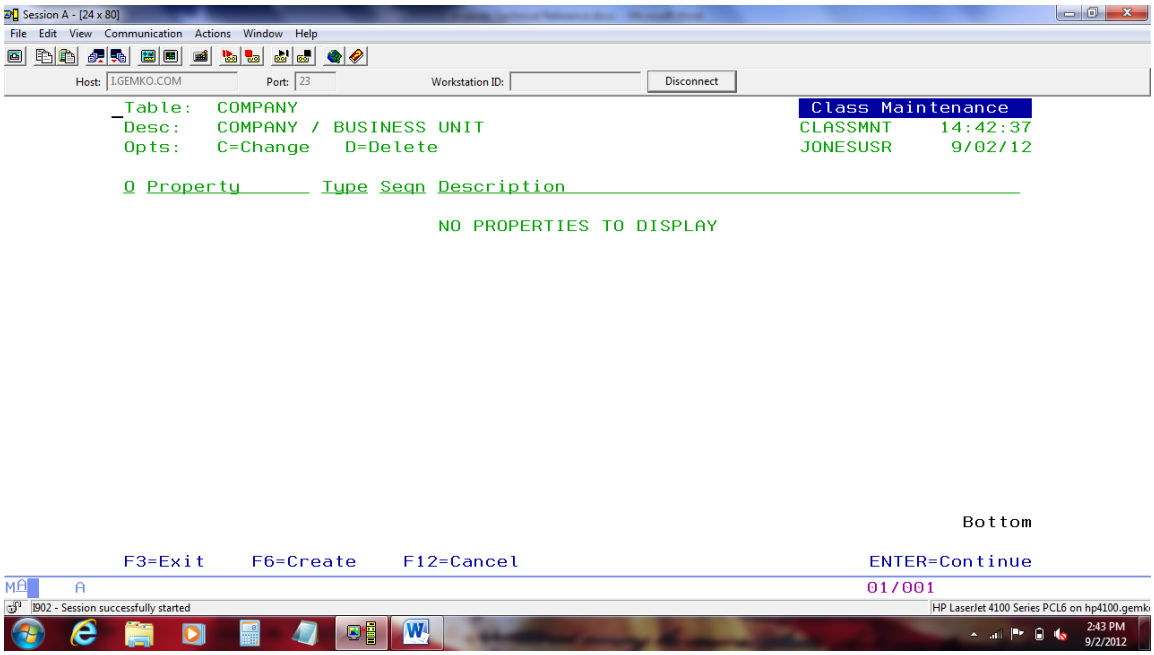


Class maintenance utility (cont.)

The new Class displays in the list. To assign Properties to it, specify the “P” option and press Enter:

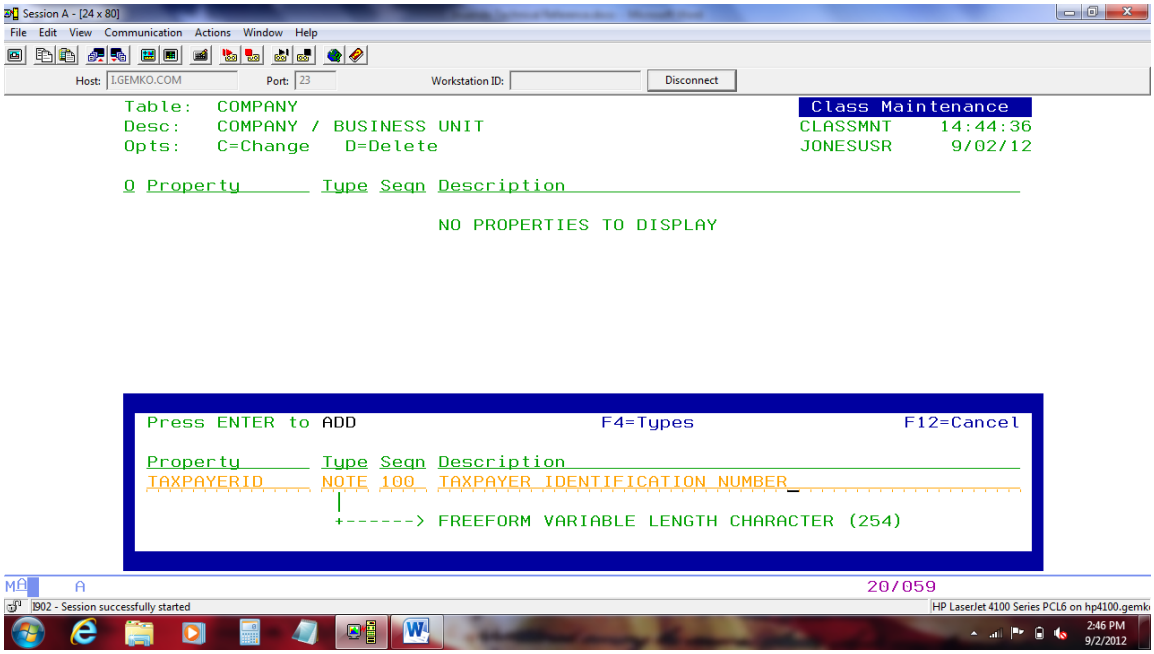


The list of Properties for the Class is shown. Press F6 to create a new Property:

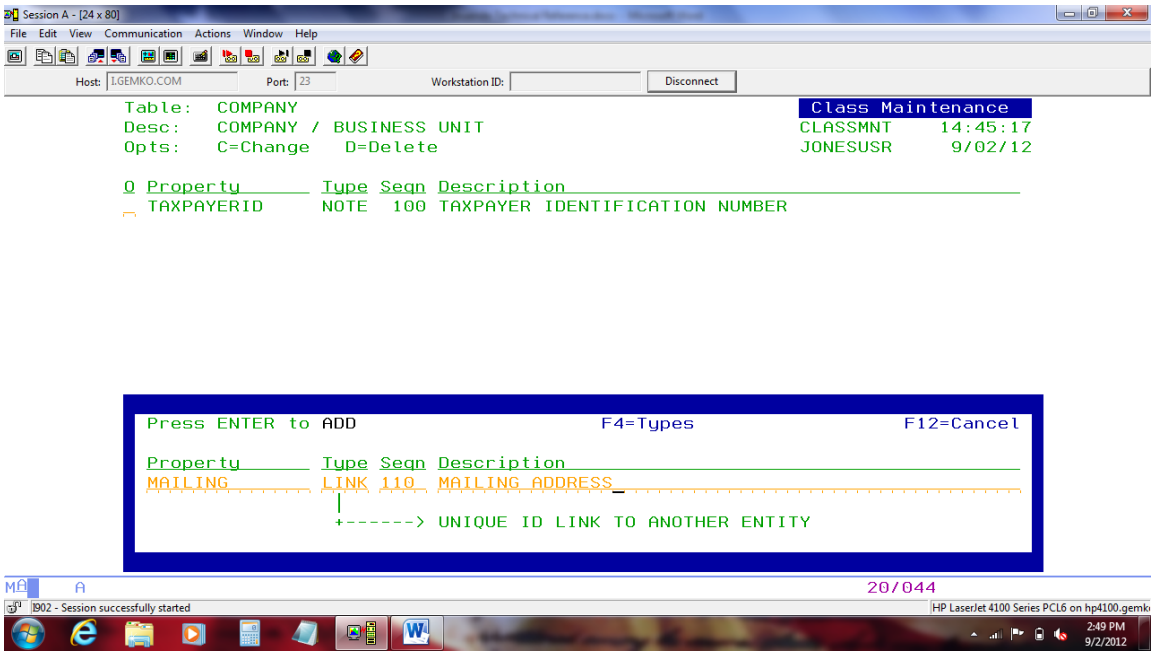


Class maintenance utility (cont.)

Key in the desired Property, Sequence and Description. Press F4 to select the desired Data Type on a rotating basis. Press Enter when satisfied:

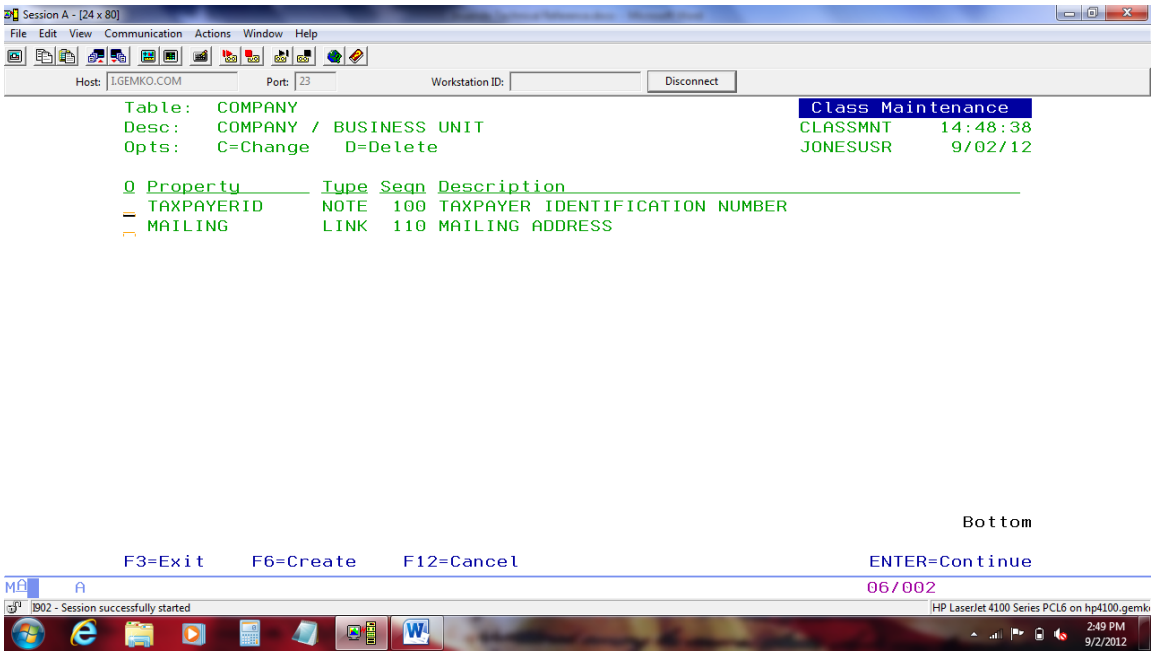


Specify additional Properties as desired. Note how MAILING is of type LINK. That means its value is an Entity ID. This example seems to suggest another class, perhaps called ADDRESS, containing typical address info:

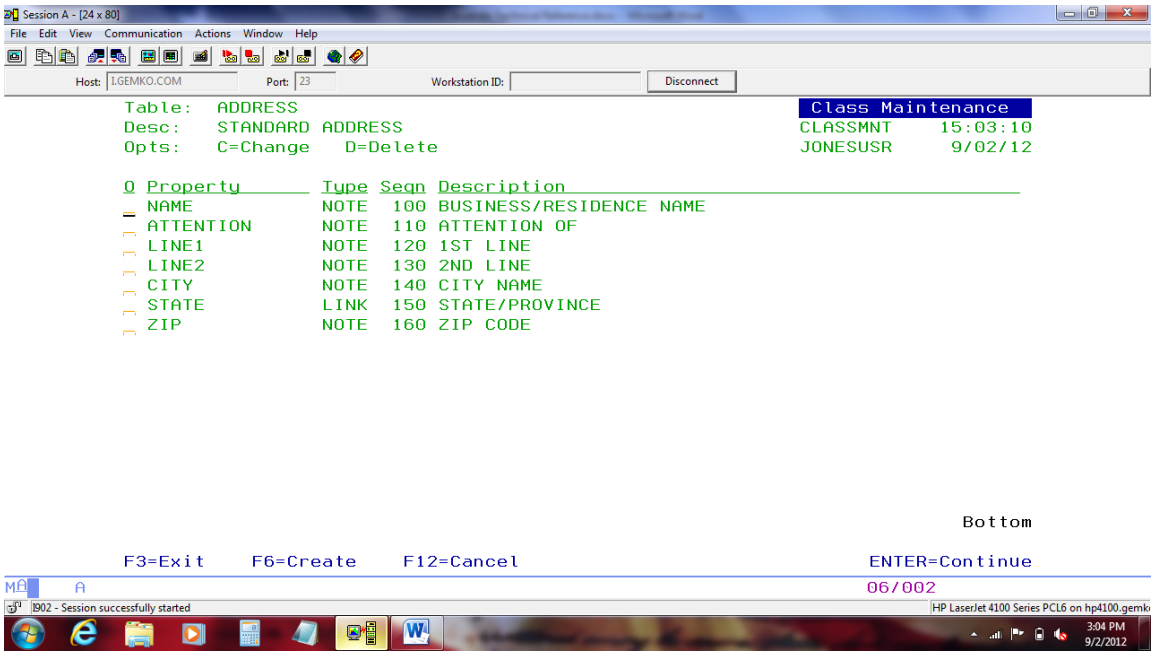


Class maintenance utility (cont.)

The list of Properties for the COMPANY Class now shows both:



Here is an example of what an ADDRESS Class might look like:



8. Time Travel Support

Inuendo **compliant** applications may travel backwards in time to a user specified Moment, and remain frozen there. The application will see the data exactly as it would have been at that Moment, however it cannot update the data while in the past.

A compliant application is defined as one which uses exclusively the Inuendo standard I/O functions for:

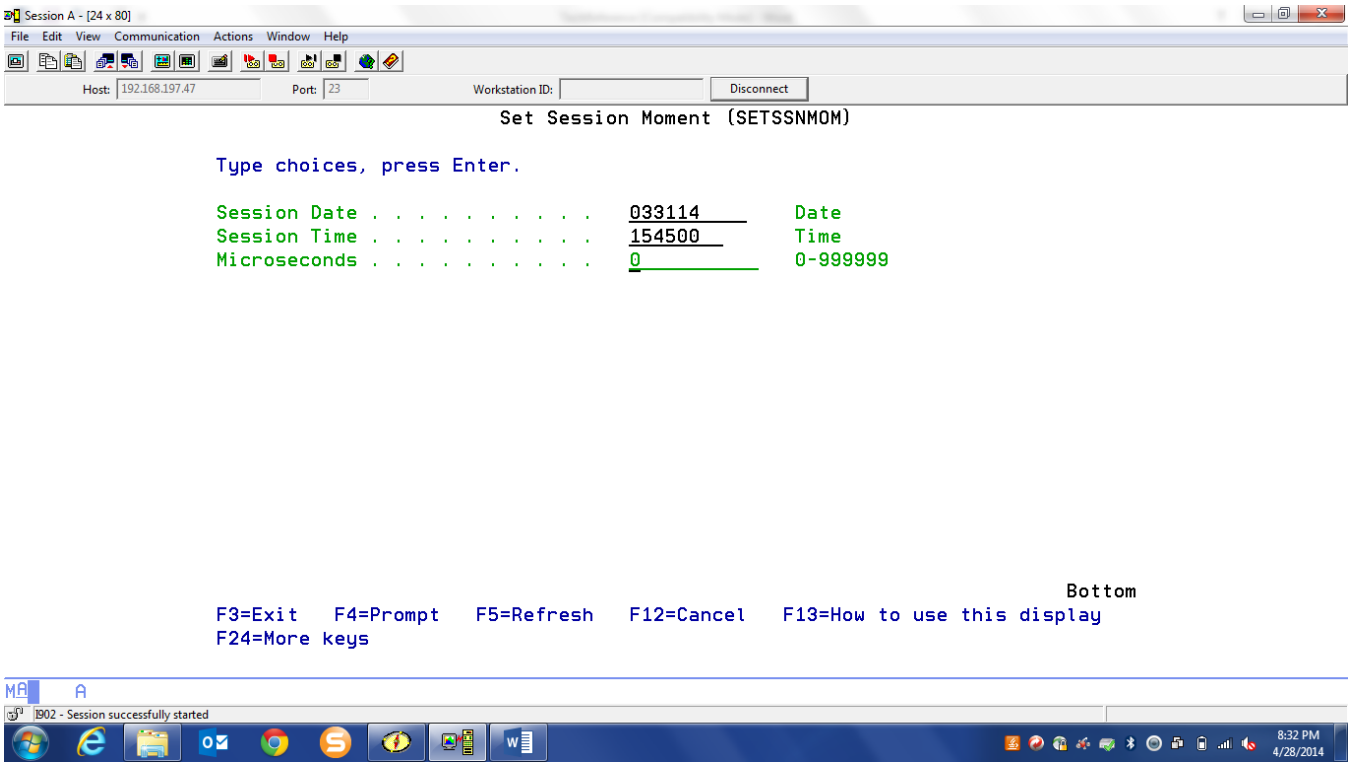
- All entity creation
- All property value setting (PUT functions)
- All identity resolution
- All property value retrieval (GET functions)

Three operating system commands have been added:

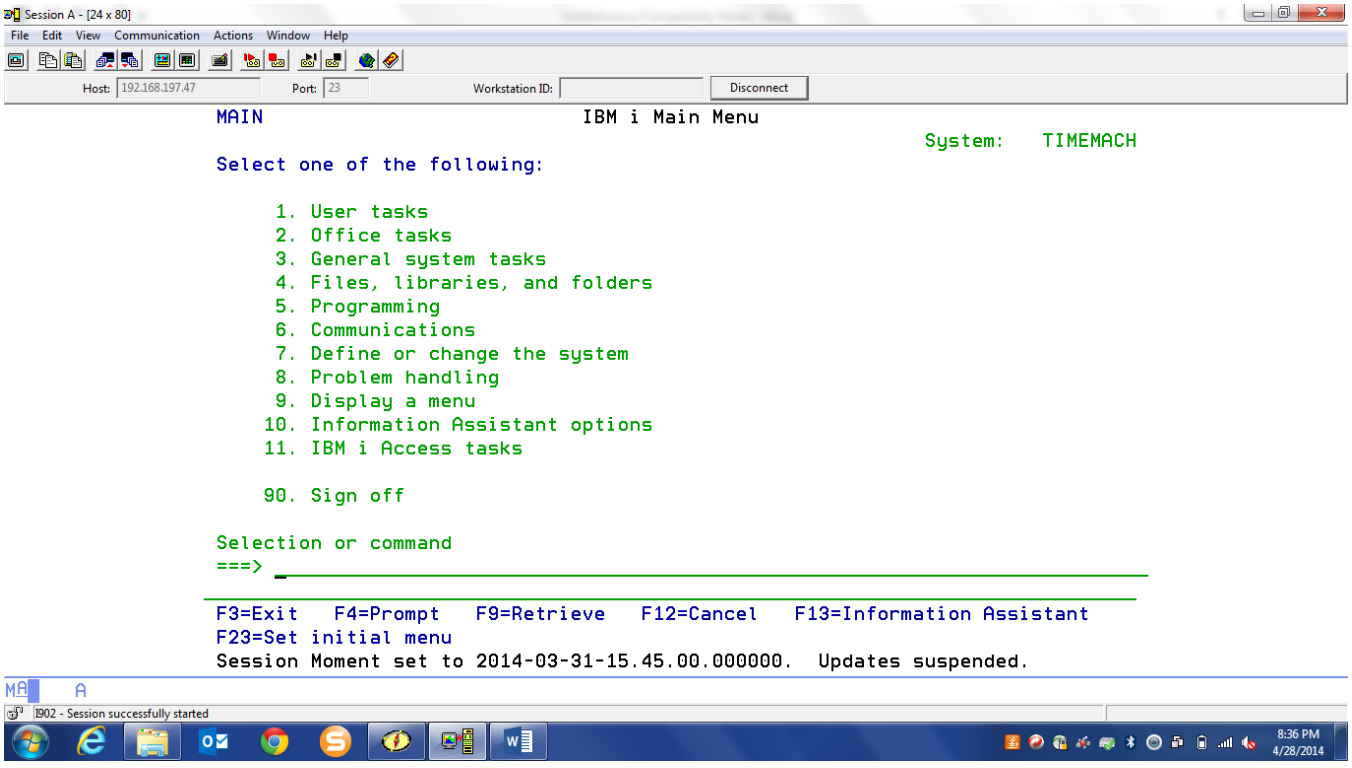
- **SETSSNMOM** (Set Session Moment) – Accepts a Moment argument in three pieces (date, time, microseconds) and records that Moment in a special user space in the QTEMP library. All native Inuendo output operations are automatically disabled, and all native Inuendo input operations will use this Moment in place of the system time stamp if no explicit Moment is specified as an argument of an Inuendo function call. Note that a subsequent issuing of this command will simply replace the contents of the user space with the newly specified Moment.
- **DSPSSNMOM** (Display Session Moment) – Echoes the Session Moment if one is in effect, in the program message line of the 5250 interactive display.
- **CLRSSNMOM** (Clear Session Moment) – Deletes the special user space in QTEMP, thereby re-enabling all native Inuendo standard output operations. All native Inuendo input operations will then use the system time stamp if no explicit Moment is specified as an argument of an Inuendo function call.

IMPORTANT: Do not tamper with user space QTEMP/SETSSNMOM once it has been created. Use only the above commands to manage the Session Moment. Otherwise undesired changes could be recorded in the database.

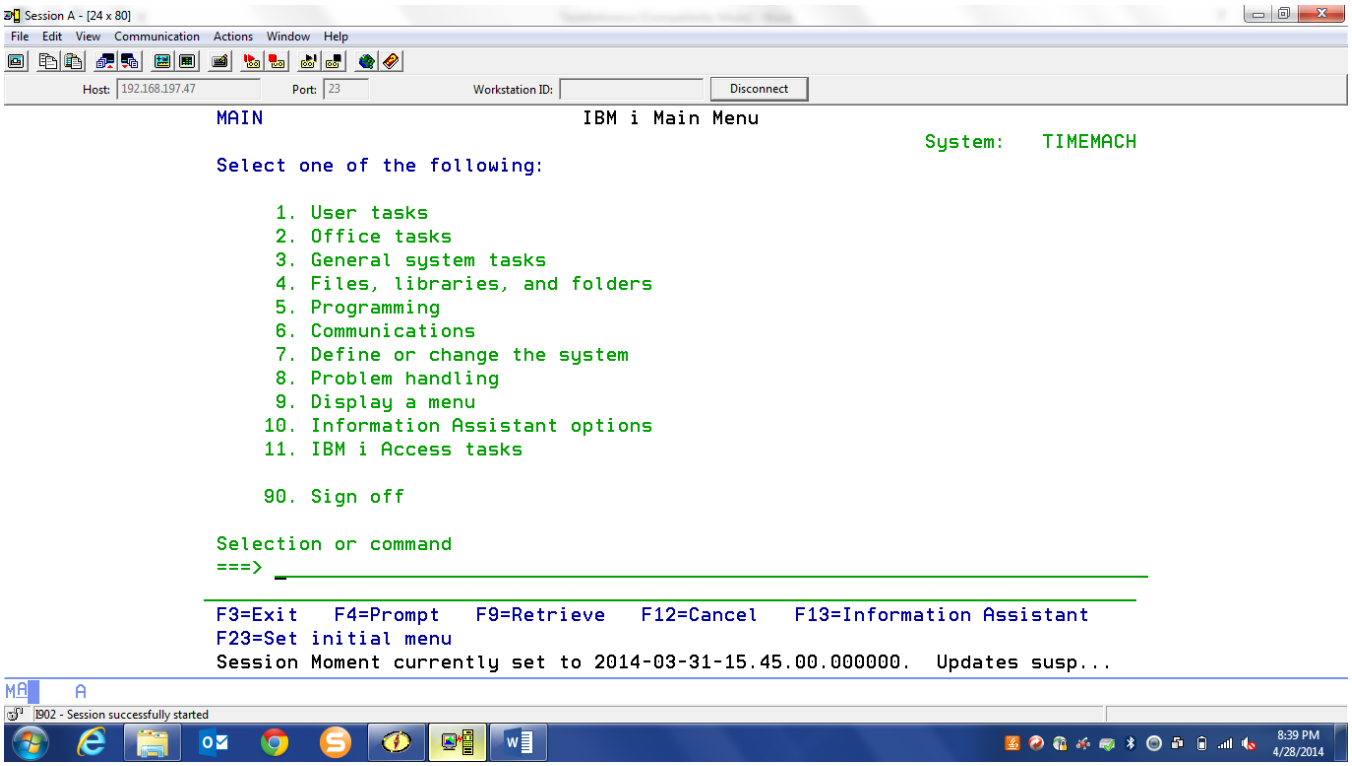
This example of SETSSNMOM would set the Session Moment to 2014-03-31-15.45.00.000000:



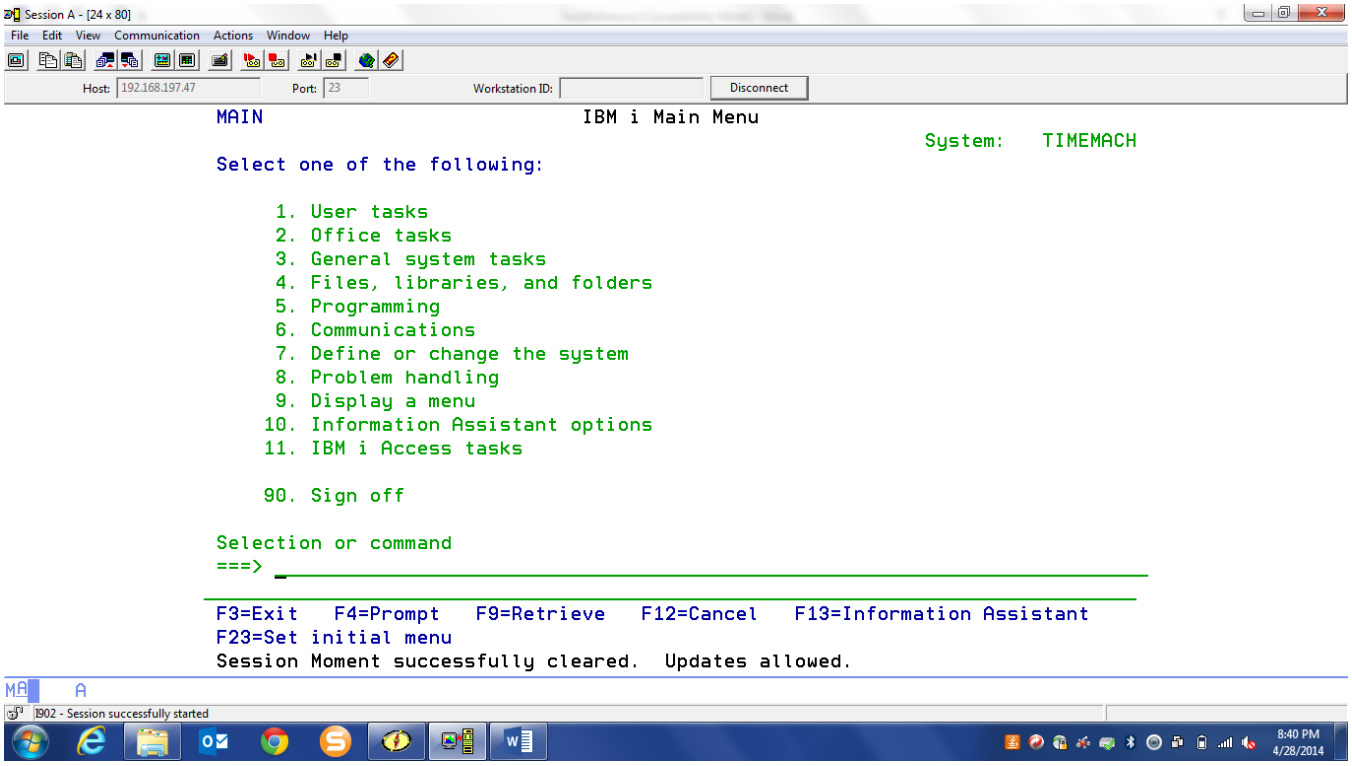
The Session Moment is echoed upon completion of the SETSSNMOM command.



The DSPSSNMOM has no parameters. It displays a similar echo screen to that of SETSSNMOM.



CLRSSNMOM take no parameters. It verifies that the Session Moment has been cleared.



9. GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights from Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation

into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's “contributor version”.

A contributor's “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.