VistA Gateway “Primer” for DataBridge

Overview: The DSS VistA Gateway is a utility that allows developers to integrate their host applications into the Department of Veterans’ Affairs (VA) legacy database: VistA. The DSS VistAGateway is based on “COM”. The DSS VistAGateway possesses many properties and methods that allow the DataBridge developers to retrieve data out of VistA and input data into the VistA. The DataBridge project allows a CIS vendor to manage the below mentioned VistA data elements in an on-demand, real time basis using the VistA Gateway COM based methods and/or properties to supplement HL7 data feeds.

• User Authentication (Security)

• Allergies

• Coding Status (Patient Care Encounter)

• Lab Orders

• Patient Selection

• Patient Registration

• Patient Record Flags

• Pharmacy Orders

• Problem List

• TIU (Progress Notes)

• VistA Imaging

• Radiology Orders

• Vitals Management

• Miscellaneous Supporting APIs

The intent of this manual is not to replace the existing VistA Gateway Programming API guide; this manual is meant to act as a primer to accompany the VistA Gateway Programming API guide for CIS DataBridge developers. Before reading this manual, please verify you have the following components in place: the VistA Gateway dll, access to the VistA Gateway Programming API guide, the VistA Gateway tester and source code, and connection to a VistA database supplied to you by DSS running in a Cache environment. If you do not have one of the above components in place, stop! Call Document Storage Systems at 561-227-

0207 and ask for the DataBridge Project Manager. It’s assumed your company has a

DataBridge contract in place with Document Storage Systems.

The remaining part of this document will cross reference the APIs detailed within VistA Gateway Programming API guide. The purpose of this manual is to assist you (the DataBridge API developer) to understand (1) the relevant API methods, (2) the object properties, (3) the interdependencies of the APIs in question, and (4) the purpose and best practice implementation for the API(s) in question. This document is not intended as a replacement for the VistA Gateway Programming API guide.

VistA Gateway User Authentication Implementation

The DataBridge applications must connect to VistA, maintain that persistent connection for further data retrieval/processing, and authenticate the user in question as a valid and legal VistA user. The VistA Gateway allows the CIS software to access VistA in the same manner as CPRS, BCMA, and other RPC broker based applications. Once an authenticated token is returned to the integrated VistA Gateway system, the software can assess object properties to store within the offline database in question and use VistA “pointer” information for further VistA package processing later on the software session. There are several “touchpoints” within VistA that require VistA user review, validation and interaction.

It’s because of these rules based processes the VistA Gateway COM object is recommended to be implemented within the host application software. The outcome is a persistent connection to the VistA database so on-demand data can be retrieved and data prompts to induce the user into determining certain data elements to be sent back into VistA; abiding by inherit VistA module rules in a real time interactive session.

Relevant Methods: To connect to VistA, the VistA Gateway developer must call the “MISC\_ConnectToVistA” API. This API will either return a “0” or a number greater than “0” which represents the internal user number: UserDUZ. Once authenticated the VistA

Gateway will maintain the connection to VistA via a TCP/IP socket; to disconnect from VistA

the “MISC\_DisconnectFromVistA” API must be called.

VistA Security: The manner that VistA allows users to invoke certain APIs (“RPC” or Remote Procedure Calls - normally known to SQL programmers as “stored procedures”) is through a VistA object called a “menu option”. VistA menu options are containers for all RPCs that a user would be allowed to invoked within a given VistA aware application. This limits VistA functionality of the application to those calls that are contained within the menu option that is associated with that application; which in turn is assigned to users of the application. This concept is referred to as “assigning a menu option to a VistA user”. Before the user can successfully connect to VistA, the VistA user administrator must be provided a list of all users that are planned to use the application. The VistA user administrator will assign the menu option in question. Upon connection to VistA, VistA will determine if the user is assigned the proper menu option and will proceed or disconnect as appropriate. The menu option is a parameter of the

“MISC\_ConnectToVistA” API. A list of menu options for each DataBridge partner is listed within Attachment C of this document. Each partner is responsible for alerting Document Storage Systems of any VistA Gateway method used within their application so the proper RPCs can be added to their associated menu option.

Relevant Object Properties: At any time during the session (providing the VistA Gateway object has been property created), the VistA Gateway developer can access any of the object properties: Object.Property. All object properties are read only. At any time during the session, the developer can verify whether or not the VistA Gateway object is connected to VistA by accessing the Object.ConnectedToVistA property; this property will return the above mentioned TCP/IP socket number (“0” if in a disconnected state). Other

useful object properties include: object.username (name of the VistA user), object.userduz (internal number of the VistA user – used within the TIU note call), object.userclass (classification of the user – nurse, doctor, clerk, etc..), object.defaultlocationIFN (internal location number for the TIU note), and object. defaultlocationname (VistA location name for display purposes). These object data elements are useful for further API calling or for user interface display.

Interdependencies: As mentioned above the VistA Gateway object must be created before any object methods and properties are accessed. The “MISC\_ConnectToVistA” and “MISC\_DisconnectFromVistA” APIs are inter-related and used to connect and disconnect the user to/from VistA. The Object.ConnectedToVistA property can be accessed at any time to determine if the above mentioned method to determine the state of connection. There must be an active connection at the time an APIs is invoked; there are a few exceptions to this rule. These “offline” APIs can be determined by reviewing the remote procedure call reference within the API manual; if nil the API is not associated with a RPC is strictly a “helper” function.

Best Practices: Users do not care for duel logon screens; one for VistA, one for SQL. VistA is a very secure government owned database inherit with all accepted security based connection rules. For this reason, it is recommended that the only log on screen that is presented to the user be the VistA Gateway invoked log on screen. This log on method will return a 0 or non-0 return. If a non-0 return is acquired the VistA Gateway developer can assume this user is authenticated as a “DataBridge” user. The return of the connection screen can then be used to compare against the SQL user table, if the SQL user table does not indicate that this user number is a legal SQL user, the system could then return a meaningful result to the user; if the user is a legal SQL table user, the user would than be connected/authenticated to the SQL and VistA database. This means a “VistA UserDUZ” field must be introduced into the native SQL database for storage and authentication.

Securing the SQL Data: The VA is very security minded. A variety of database management and reporting tools are available to various VA employees that are not necessarily allowed to access DataBridge data. These database tools could allow unauthorized VA employees access to SQL data; many off line SQL databases’ store VistA “personally identifiable information”. Because of the inherit nature of SQL databases “wanting to be connected to”; security measures must be taken into consideration before implementing an SQL database within a VA environment. There are many technical approaches one can take to “secure” their SQL database ranging from encryption to creating a middle tier such as MTS. The VA will require an attempt on your part to secure your SQL database against unauthorized user access and/or database tool access. A “PIA” (Privacy Impact Assessment) will be

performed during your pilot implementation; you will not be allowed to move into secondary VA sites without document how you secure your SQL data from unauthorized access. A “PIA” is available upon request for review and will be discussed with you by the Document Storage Systems DataBridge Product Line Manager to determine how vulnerable your offline data is. The VA considers VistA

the primary “system of records” and requires any vendor data repository to be secure, and document what kind of VistA data the data repository is storing.

Maintaining a secondary “System of Records” within your SQL database: Special consideration must be given to VistA information that is stored within your SQL database. VistA is considered the primary system of records for patient, user, associated clinical information. By nature of the integration itself to VistA, the CIS system will need to store a reasonable amount of duplicate VistA data if for no other reason than for further VistA data processing and/or for user interface data display. Ideally, the CIS database should store the minimum amount of VistA data in the forms of internal entry numbers (known as “pointers”). These internal entry numbers (IENs) could then be used to acquire the “personally identifiable information” that is stored within the primary system of records: VistA. Practically, this most like will not occur as the CIS system is treated as an extension of the VistA system’s clinical information. CIS systems are encouraged to minimize duplicate “personally identifiable information” storage; and when it’s necessary to store the “personally identifiable information” document what VistA information is stored and for what reason so PIA applications can be completed as efficiently as possible.

VistA Gateway Allergy Implementation

After the CIS system connects to VistA through the VistA Gateway, a variety of method calls are available to the VistA Gateway developers to process and display clinical

information to the user. Two such method calls give the VistA Gateway developer the ability to retrieve a list of patient allergies from VistA as well as retrieving the details about a given allergy.

Relevant Methods: Assuming an active connection is established, the VistA Gateway developer must call the “CPRS\_GetPatientAllergyList” API. This API will either return a “-

1^Message” or a list of allergies and their associated internal numbers. This list view might be sufficient; if not, the user can drill down into the details of each allergy using the “CPRS\_GetAllergyDetails” API (providing the system has access to the allergy internal number).

Relevant Object Properties: The allergy APIs are patient specific, this means the VistA patient internal number (IFN) will need to be available to retrieve the allergy list. Normally the patient number will most likely be stored within the CIS database. If so, when patient context changes within the CIS system, the patient number should be available to send into the above mentioned “CPRS\_GetPatientAllergyList” API method. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of this document) should be called to set the object patient properties. The object stores numerous patient demographic data elements that can either be displayed to the user, create a new patient record, or update

the HL7 ADT information stored in the offline SQL CIS database.

Interdependencies: A patient internal number must be available to acquire the patient allergy list. The allergy internal number is needed to acquire the details about the allergy.

Best Practices: The two allergy APIs have a master/detail relationship. Normally, the CIS system has pre-existing screens that display clinical master/detail information via a tree- view object that displays the master as a parent node and the details and children notes associated with the parent. Another way to display this information is to implement a list box that displays the allergy list and when an allergy is selected an associated text box displays the details.

VistA Gateway Coding Status Implementation

The primary coding interface(s) is HL7 based. The appointments and checked out visits will be sent to the CIS through a SIU HL7 outbound from VistA into the CIS; this will allow the CIS to send the associated coding event back into VistA. There will be an in-place VistA based HL7 inbound listener that will accept and acknowledge the CIS coding event HL7 messages. Two HL7 VistA based “application” components are in place to direct coding messages from the CIS to specified ‘IN” protocols to a processing routine (EN^DSIHIN) that will digest the coding messages. Once the routine has parsed the message into local arrays, the data enclosed in the messages will be stored in PCE using the DATA2PCE^PXAPI M based API. For more details on the HL7 DataBridge coding interface, please be referred to the DataBridge HL7 Specification; particularly the SIU and DFT sections.

As an alternative to the HL7 coding event DataBridge interfaces, the VistA Gateway has encapsulated much of the visit/appointment retrieval events as well as coding update events through a variety of COM APIs. These APIs will allow the CIS to retrieve a patient based appointment/visit listing for a particular date range and then take coding action on a given visit.

Relevant Methods: Before coding events can occurs through an API, a list of visits with corresponding visit numbers needs to be introduced within the CIS. The “PCE\_ApptListByPatient” API will allow the VistA Gateway developer to acquire a list of appointments and existing visits for a given patient by date range. If the appointment has an associated visit (a checked-in/checked-out appoint), a visit number (“VisitIFN”) will be associated with it. The “PCE\_RetrieveVisitDetails” API will allow the VistA Gateway developer the ability to determine what existing ICD/CPT codes are associated with that visit. This API is particularly useful if the CIS in question has certain coding rules embedded into the software. This API will allow the CIS to display and process certain ICD/CPT codes back into VistA without unintentionally duplicating existing codes. The “PCE\_AddICDCodeToVisit” and “PCE\_AddCPTCodeToVisit” APIs allow the CIS to identify certain ICD and CPT codes within the CIS and send those codes back into VistA for a pre- designated associated visit. There are also “cousin” APIs that will allow the CIS to identify pre-existing ICD and CPT codes that are currently associated with the visit and delete them, they are: “PCE\_DeleteCPTEntryFromVisit” and “PCE\_DeleteICDEntryFromVisit”.

Relevant Object Properties: The visit management APIs are patient specific, this means the VistA patient internal number (IFN) will need to be available to retrieve the visit list. Normally the patient number will most likely be stored within the CIS database. If so, when patient context changes within the CIS system, the patient number should be available to send into the above mentioned “PCE\_ ApptListByPatient” API method. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of this document) should be called to set the object patient properties. After the VistA Gateway successfully renders a visit list to the CIS, the associated visit number (“VisitIFN”) can be retrieved for further visit/coding processes.

Interdependencies: A patient internal number must be available to acquire the patient visit list. The visit list array will contain the visit numbers for the associated visit in question. The Visit number will allow the CIS to drill down into the details about the visit and furthermore allow the CIS to send coding updates back into VistA using the “PCE\_AddICDCodeToVisit” and “PCE\_AddCPTCodeToVisit” APIs; which are visit number centric.

Best Practices: The CIS should primarily collect visit information via the DataBridge HL7 event drivers; SIU messages. This means the CIS database must be designed to be “VistA relevant”. The CIS database should have existing fields within it’s patient and visit/appointment database tables to minimally store host “entry pointers” or associated patient and visit numbers that are relevant to the host system in question: VistA. This means the CIS database must accommodate for a “VistA Patient Number” and “VistA visit number” field value(s) within each of it’s patient and visit based SQL tables. Once the CIS database is made to be “VistA relevant”, coding event data processing can occur. As stated above, the recommended practice is to utilize existing HL7 events for appointment retrieval and coding event updating. However, there are times that new patients get introduced into the CIS through CCOW patient context changes. It is for these “on-demand” unplanned events that the VistA Gateway COM object APIs are most useful. The above APIs should be implemented by the CIS if the patient in question is detected to be a new patient introduced into the CIS. At that time, historical visits the to CIS will be unknown and the visit

retrieval API could be invoked to seed the CIS visit table for a given date range. One other occurrence that for “on demand” visit retrieval should be implemented is for visit retrieval outside the boundary of CIS implementation. Many times, the CIS is implemented and the user desires to review historical visits outside the HL7 implementation date range. With COM object implementation visit data retrieval can supplement HL7 data and guarantee the user will be able to view any patient and visit related data element without having to be concerned with cut over dates.

VistA Gateway Lab Ordering Information Implementation

The primary lab data retrieval interface(s) is HL7 based. The lab data will be sent to the

CIS through an ORM HL7 outbound from VistA into the CIS. For more details on the HL7

DataBridge lab interface, please be referred to the DataBridge HL7 Specification;

particularly the ORM section.

As an alternative to the HL7 lab event(s) DataBridge interfaces, the VistA Gateway has encapsulated much of the lab results retrieval events through a variety of COM APIs. After the CIS system connects to VistA through the VistA Gateway, a variety of method calls are available to the VistA Gateway developers to process and display clinical information to the user. Two such method calls give the VistA Gateway developer the ability to retrieve a list

of patient lab orders from VistA as well as retrieving the details about a given lab order. These APIs will allow the CIS to retrieve patient based lab results for a particular date range.

Relevant Methods: Assuming an active connection is established, the VistA Gateway developer must call the “CPRS\_GetPatientLabList” API. This API will either return a “-

1^Message” or a list of patient lab orders and their associated internal numbers with the following status’: "DISCONTINUED", "COMPLETE", "HOLD", "FLAGGED", "PENDING", "ACTIVE", "EXPIRED", "SCHEDULED", "PARTIAL RESULTS", "DELAYED", "UNRELEASED", “LAPSED”, “RENEWED”, “CANCELLED“, “DISCONTINUED/EDIT", "NO STATUS". This list view might be sufficient; if not, the user can drill down into the details of each lab order list entry using the “CPRS\_GetPatientLabDetails” API (providing the system has access to the

lab order internal number). If a “CUM” (cumulative) report is desired the “CPRS\_GetPatientLabCum” API is available; the “CPRS\_GetPatientLabInterim” API is the cousin call that will allow the VistA Gateway developer to display the interim lab results for a given date range.

Relevant Object Properties: The lab APIs are patient specific, this means the VistA patient internal number (IFN) will need to be available to retrieve the lab order(s) listing. Normally the patient number will most likely be stored within the CIS database. If so, when patient context changes within the CIS system, the patient number should be available to send into the above mentioned “CPRS\_GetPatientLabList” API method. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of this document) should be called to set the object patient properties. The object stores numerous patient demographic data elements that can either be displayed to the user, create a new patient record, or update the HL7 ADT information stored in the offline SQL CIS database.

Interdependencies: A patient internal number must be available to acquire the patient lab order(s) listing. The lab order internal number is needed to acquire the details about the lab order.

Best Practices: The lab APIs have a master/detail relationship. Normally, the CIS system has pre-existing screens that display clinical master/detail information via a tree-view object that displays the master as a parent node and the details and children notes associated with the parent. Another way to display this information is to implement a list box that displays the lab order listing and when a lab order is selected an associated text box displays the details.

VistA Gateway Patient Selection Implementation

After the CIS system connects to VistA through the VistA Gateway, a patient context change can occur through a variety of different means. There are “silent” APIs to assist in inherit patient context changes (CCOW, native SQL patient search dialog) as well as interactive APIs that allow the VistA Gateway developer to invoke a VistA Patient Search/Select interactive dialog box. The primary methodology to retrieve patient related data from VistA will be through native DataBridge ADT HL7 feeds. These HL7 feeds will seed the initial patient records within the CIS SQL database. There several existing VistA Gateway methods in place to search for specific VistA patients in real time as the session demands. The VistA Gateway can allow the CIS system to verify a CCOW message and retrieve necessary patient related information to seed a new record on the fly, if the existing CIS database does not have that patient on file via historical HL7 data.

Relevant Methods: The VistA Gateway offers real time patient lookup APIs. These APIs are categorized into user invoked interactive APIs and silent data processing APIs. The VistA Gateway developer can introduce a real time VistA patient selection dialog box through the implementation of the “PAT\_SearchForVistAPatient” API. Many times the CIS already has patient search interactive dialogs set up within the pre-existing user interface and what is required is a set of “silent” APIs that allows the CIS to verify or lookup a VistA patient using key patient indicators/indexes. The main API that can be called silently to verify VistA patient existence and associated data is the “PAT\_SetNewPatient” API. This API will return a Boolean; if true, the VistA Gateway object properties will be reset to the newly desired VistA patient. If the VistA Gateway developer does not want to reset the object properties and is desiring query results the “PAT\_GetDemographics” API would be appropriate.

Relevant Object Properties: Many of the VistA Gateway patient specific object properties are related to the above stated methods. The “PAT\_SearchForVistAPatient” and “PAT\_SetNewPatient” API results are a change in the patient specific object properties.

The patient specific object property values are the basic demographic values that are stored within the VistA patient file (#2) as of that moment in time. The “PAT\_GetDemographics” API will not update the object properties.

Interdependencies: A patient internal number must be available to acquire the patient demographic/object property values. Normally, the patient internal number (“PatientIFN”) will be acquired from the PID segment of the ADT HL7 messages.

Best Practices: The primary mechanisms to seed VistA patient related information within the CIS should be the ADT HL7 DataBridge feeds. These data feeds will seed the CIS SQL database. At times, new patients will be introduced into the CIS through unknown CCOW messages or on-demand user events. For these occurrences, the VistA Gateway object should be used. The VistA Gateway object is an excellent “fall back” to acquire VistA

patient related data in a query based mode. The CIS can query the VistA Gateway object through the above “silent” data processing API to initial set up a patient record that might

be unknown to the CIS SQL database. The CIS can also introduce a real time patient search and selection dialog box that will return the selected patient and associated patient demographic information to the CIS.

VistA Gateway Patient Registration Implementation

Please refer to the above stated “PAT\_SetNewPatient” and “PAT\_GetDemographics” API within the patient selection section of this document. After the above patient search APIs are invoked the following object property values will be available to the CIS to introduce into their offline patient table (all string values):

PatientIFN: A number that reflects the patient’s internal file number. PatientName: 3-30 upper-case characters in length, Last,First Middle. SensitivePatient: 0=Not a sensitive patient, 1=Sensitive patient. PatientGender: ‘M’ for male, ‘F’ for female.

PatientDOB: MM/DD/YYYY, could be imprecise i.e. 00/DD/YYYY PatientAge: TCalculation based on the above filed DOB i.e. 65. PatientMaritalStatus: See set of codes below.

PatientRace: See set of codes below.

PatientReligion: See set of codes below.

PatientSSN: 9 (xxxxxxxxx) characters in length, or 10 (xxxxxxxxxP) in length for a pseudo-SSN.

PatientAddress1: 3-35 characters in length. PatientAddress2: 3-30 characters in length. PatientAddress3: 3-30 characters in length. PatientCity: 3-15 characters in length. PatientCounty: 3-15 characters in length. PatientState: 2 upper-case characters in length. PatientZip: 5 characters in length. PatientHomePhone: 4-20 characters in length. PatientWorkPhone: 4-20 characters in length. PatientEmploymentStatus: See set of codes below.

PatientDeathDate: MM/DD/YYYY, could be imprecise i.e. 00/DD/YYYY.

Example:

711^CPRSPATIENT,EIGHT F^666660008;666-66-

0008^2550201;02/01/1955^52^F^;^WHITE, NOT OF HISPANIC ORIGIN^LUTHERAN^MARRIED^EMPLOYED PART TIME^22233 TEST LANE TEST LANE TEST LAZY^^^ALBANY^NY^12211;^ALBANY;^555-2222^555-

8888^EIGHT;CPRSPATIENT;F;

Set of codes:

1Race – Possible values can vary by site. Examples: AMERICAN INDIAN OR ALASKA NATI

ASIAN

BLACK OR AFRICAN AMERICAN DECLINED TO ANSWER

NATIVE HAWAIIAN OR OTHER PACIF UNKNOWN BY PATIENT

WHITE

2Religion – Possible values can vary by site. Examples: ADVENTIST

ASSEMBLY OF GOD BAPTIST BRETHREN BUDDHIST CATHOLIC

CHRISTIAN SCIENTIST CHURCH OF CHRIST CHURCH OF GOD DISCIPLES OF CHRIST EASTERN ORTHODOX EPISCOPALIAN EVANGELICAL COVENANT FRIENDS

ISLAM

JEHOVAH'S WITNESS JEWISH

LATTER-DAY SAINTS LUTHERAN METHODIST

NATIVE AMERICAN NAZARENE

OTHER PENTECOSTAL PRESBYTERIAN

PROTESTANT, NO DENOMINATION PROTESTANT, OTHER

REFORMED SALVATION ARMY UNITARIAN; UNIVERSALIST UNITED CHURCH OF CHRIST UNKNOWN/NO PREFERENCE

3MaritalStatus – Possible values are: DIVORCED

MARRIED

NEVER MARRIED SEPARATED UNKNOWN WIDOW/WIDOWER

4EmploymentStatus – Possible values are: EMPLOYED FULL TIME

EMPLOYED PART TIME

NOT EMPLOYED SELF EMPLOYED FOR RETIRED

FOR ACTIVE MILITARY DUTY UNKNOWN

VistA Gateway Patient Record Flags Implementation

After the CIS system connects to VistA through the VistA Gateway, a variety of method calls are available to the VistA Gateway developers to process and display clinical

information to the user. Two such method calls give the VistA Gateway developer the ability to retrieve a list of patient record flags from VistA as well as retrieving the details about a given patient record flag. Patient record flags within VistA alert the user to special circumstances that involve a particular patient; such “flags” could be “behavioral” as one example.

Relevant Methods: Assuming an active connection is established, the VistA Gateway developer must call the “CPRS\_GetPatientFlagListing” API. This API will either return a “-

1^Message” or a list of patient record flags and their associated ID. This list view might be sufficient; if not, the user can drill down into the details of each patient record flag using the “CPRS\_GetPatientFlagReport” API (providing the system has access to the patient record flag ID).

Relevant Object Properties: The patient record flag APIs are patient specific; this means the VistA patient internal number (IFN) will need to be available to retrieve the patient record flag list. Normally the patient number will most likely be stored within the CIS database. If so, when patient context changes within the CIS system, the patient number should be available to send into the above mentioned “CPRS\_GetPatientFlagListing” API method. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of this document) should be called to set the object patient properties. The object stores numerous patient demographic data elements that can either be displayed to the user, create a new patient record, or update the HL7 ADT information stored in the offline SQL CIS database.

Interdependencies: A patient internal number must be available to acquire the patient record flag list. The patient record flag ID is needed to acquire the details about the patient record flag.

Best Practices: The two patient record flag APIs have a master/detail relationship. Normally, the CIS system has pre-existing screens that display clinical master/detail information via a tree-view object that displays the master as a parent node and the details and children notes associated with the parent. Another way to display this information is to implement a list box that displays the patient record flag list and when a patient record flag is selected an associated text box displays the details

VistA Gateway Pharmacy Ordering Information Implementation

The primary medication data retrieval interface(s) is HL7 based. The medication data will be sent to the CIS through an OBX HL7 outbound from VistA into the CIS. For more details on the HL7 DataBridge medication interface, please be referred to the DataBridge HL7 Specification; particularly the OBX section.

As an alternative to the HL7 pharmacy event(s) DataBridge interfaces, the VistA Gateway has encapsulated much of the medication results retrieval events through a variety of COM APIs. After the CIS system connects to VistA through the VistA Gateway, a variety of method calls are available to the VistA Gateway developers to process and display clinical information to the user. Two such method calls give the VistA Gateway developer the ability to retrieve a list of medication orders from VistA as well as retrieving the details about a given pharmacy order. These APIs will allow the CIS to retrieve patient based pharmacy ACTIVE orders and their given details.

Relevant Methods: Assuming an active connection is established, the VistA Gateway developer must call the “MED\_GetPatientMedlist” API. This API will either return a “-

1^Message” or a list of patient ACTIVE medication orders and their associated internal numbers. This list view might be sufficient; if not, the user can drill down into the details of each medication order list entry using the “CPRS\_GetPatientMedDetails” API (providing the system has access to the pharmacy order internal number).

Relevant Object Properties: The medication APIs are patient specific, this means the VistA patient internal number (IFN) will need to be available to retrieve the pharmacy order(s) listing. Normally the patient number will most likely be stored within the CIS database. If so, when patient context changes within the CIS system, the patient number should be available to send into the above mentioned “MED\_GetPatientMedlist” API method. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of

this document) should be called to set the object patient properties. The object stores numerous patient demographic data elements that can either be displayed to the user, create a new patient record, or update the HL7 ADT information stored in the offline SQL CIS database.

Interdependencies: A patient internal number must be available to acquire the patient pharmacy order(s) listing. The pharmacy order internal number is needed to acquire the details about the active pharmacy order.

Best Practices: The pharmacy APIs have a master/detail relationship. Normally, the CIS system has pre-existing screens that display clinical master/detail information via a tree- view object that displays the master as a parent node and the details and children notes associated with the parent. Another way to display this information is to implement a list box that displays the pharmacy order listing and when a pharmacy order is selected an associated text box displays the details.

VistA Gateway Problem List Implementation

After the CIS system connects to VistA through the VistA Gateway, a variety of method calls are available to the VistA Gateway developers to process and display clinical

information to the user. Two such method calls give the VistA Gateway developer the ability to retrieve a list of patient problems from VistA as well as retrieving the details about a given problem list entry.

Relevant Methods: Assuming an active connection is established, the VistA Gateway developer must call the “CPRS\_GetPatientProblemList” API. This API will either return a “-

1^Message” or a list of patient problems and their associated internal numbers. This list view might be sufficient; if not, the user can drill down into the details of each problem list entry using the “CPRS\_GetProblemDetails” API (providing the system has access to the problem list internal number).

Relevant Object Properties: The problem list APIs are patient specific, this means the VistA patient internal number (IFN) will need to be available to retrieve the patient problem list. Normally the patient number will most likely be stored within the CIS database. If so, when patient context changes within the CIS system, the patient number should be available to send into the above mentioned “CPRS\_GetPatientProblemList” API method. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of this document) should be called to set the object patient properties. The object stores numerous patient demographic data elements that can either be displayed to the user, create a new patient record, or update the HL7 ADT information stored in the offline SQL CIS database.

Interdependencies: A patient internal number must be available to acquire the patient problem list. The problem list internal number is needed to acquire the details about the patient problem list entry.

Best Practices: The two problem list APIs have a master/detail relationship. Normally, the CIS system has pre-existing screens that display clinical master/detail information via a tree-view object that displays the master as a parent node and the details and children notes associated with the parent. Another way to display this information is to implement a list box that displays the patient problem list and when an problem is selected an associated text box displays the details.

VistA Gateway TIU Implementation

TIU is a module within the VistA database that would commonly be associated to other EMRs “charting notes” or “progress notes” implementation. The TIU notes package within VistA is very much “rules” oriented that at times requires immediate user feedback to process information correctly back into VistA. This package requires the user to electronically sign the note in question and verify that the electronic signature is correct at the time of filing. To create a TIU note, various supporting VistA data elements associated with the TIU note must be identified by either the system or the user. The Patient, Title, Location, Visit/Appointment, Author/User, and Note Text are required data element for a TIU note. This means, the system must collect certain pieces of information before the main TIU note creation API is called via the VistA Gateway. Some of the ancillary information is collected within other areas of the application, such as user/author data, patient data, and patient specific appointment/visit data. Past implementation of TIU note creation by other VistA Gateway developers stored the TIU title, and clinic/ward information as a default within the vendor SQL database; this information was seeded upon initialization of the system.

COM object API implementation to create TIU notes is the primary technology that will be use by the DataBridge partners.

Relevant Methods: A TIU note can be easily created given a user/system has pre-identified a number of different data elements and the user in question is an authorized TIU note creator. The note can only be created after a patient (see “Patient Selection” section), location, title, visit (see “Coding Status” section), author/user (see “User Authentication” section), and note text have been determined; see “TIU\_CreateTIURecord” API. Before

the TIU notes is sent to VistA the “TIU\_Authorization” API must be called to determine if the user in question has proper authority to edit and/or create a TIU note. Other supporting APIs that could be used to manage TIU notes after the fact include “TIU\_EditTIUUnSignedNote” (to edit the contents of a known unsigned TIU note) and “TIU\_RetrieveUnsignedNoteText” which retrieves the actual text of the note on file within VistA for further editing/updating. Other unique VistA related data element necessary for TIU note creation is VistA location identification. A VistA location would otherwise be known as a “clinic” or a “ward”. These VistA locations have an associated internal entry number (IEN) which will be used when creating a TIU note. There are various APIs that will retrieve VistA location information; some are “silent” some are interactive. The “MISC\_GetLocationInfo” API is used to retrieve details about a given location; the exact location name must be known. The “USR\_GetUserLocation” API will retrieve the default location and details about that location that a VistA user is assigned to. The “PAT\_GetCurrentPatientLocation” API will retrieve the default location and details about that location that a VistA patient is assigned to. The VistA location will need to be determine (however the workflow is to be determined within the application), before a TIU note can be created. The TIU note title is another required data element that is used to categorize TIU notes. The “TIU\_ShowTitleSearchDialog” API is an interactive dialog box that is presented to the user to determine what title shall be used to create a TIU note.

The “CPRS\_ValidateESignature” API should be used to validate whether or not the logged in user’s electronic signature code is valid within VistA. Once validated the electronic signature code string would be used against the “TIU\_SignTIUNote” API to sign off on the created TIU note.

Relevant Object Properties: The TIU note creation processes is patient specific, this means the VistA patient internal number (IFN) will need to be available to create a TIU note. The active user will be normally treated as the note author. Normally the patient and user number will most likely be stored within the CIS database. If so, when patient context changes within the CIS system, the patient number should be available to send into the above mentioned “TIU\_CreateTIURecord” API method with the logged in user number. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of

this document) should be called to set the object patient properties. The object stores numerous patient demographic data elements that can either be displayed to the user, create a new patient record, or update the HL7 ADT information stored in the offline SQL CIS database.

Interdependencies: There are several data elements that are necessary for a TIU note to be successfully created. A patient internal number, user number, location identifier, and note title must be identified before a TIU note can be created within the VistA system via the VistA Gateway object.

Best Practices: To prepare the user to send back a TIU note back to VistA, the CIS user interfaces must prepare the user to collect information critical for the creation of a TIU note. The user must be induced into choosing “ancillary” data that normally is considered supplementary data to the note in question. Mainly the Visit, Location, and note Title information. The note Visit: the visit normally had been collected by the CIS through a series of SIU HL7 messages (see the “Coding Status” section above). The TIU title: several options are available to the VistA Gateway developer. As stated above, the CIS system could present the above mentioned interactive title search dialog to induce the user into selecting a TIU title for that session or TIU note; another option is to stored the TIU title information as a default within the CIS SQL database which would be seeded upon initialization of the system. The location: several options are available to the VistA Gateway developer. As stated above, the CIS system could present the above mentioned interactive location search dialog to induce the user into selecting a ward/clinic for that

session or TIU note; another option is to stored the ward/clinic information associated with each user as a default within the CIS SQL database which would be seeded upon initialization of the system. Once the required data elements are collected within a TIU

user interface screen(s) within the CIS, the VistA Gateway API that creates a TIU note will be invoked. The CIS shall store the TIU note number for later retrieval and validation so the TIU note content and status can be displayed to the user.

VistA Gateway VistA Imaging Implementation

At times, the CIS will have the need to send CIS produced images back into VistA. The host VistA system that stores images is known as “VistA Imaging”. The VistA Imaging system provides a finite number of APIs to the VistA Gateway for imaging processing. The CIS

must associate a TIU note to each image it expects to send back to the VistA Imaging system. The outcome of this API set is an image association to a TIU note within the CPRS system; this association is graphically depicted with a picture icon to the left of each TIU note reference with the CPRS TIU note listing.

VistA Imaging is a package within the VistA database that allows images produced by a CIS to be stored as part of the patient record. The capable image types range from picture files of various formats to PDF images. Images in VistA are frequently associated with a TIU progress note to provide supporting documentation and, therefore, a more complete record.

To upload a TIU associated image to VistA Imaging, various supporting elements (parameters) must first be identified by either the system or the user. The patient, image file name and network location, image date, unique imaging identifier, and TIU note for association are all required for VistA Imaging to store the image and associate it with a TIU note.

The patient and TIU note are readily available from the creation of a TIU note described in the “TIU Implementation” section of this document. The file name and network location is simply a fully qualified server name and path to where the image file is located. This file an location must be accessible to VistA Imaging across the hospital network as VistA Imaging will be making a copy of the file to a more permanent storage location. The unique imaging identifier is described in detail in the VistA Gateway Programming API guide, but is simply a unique key for VistA Imaging to track the status of the image upload and provide feedback on that status.

Another aspect of implementing the use of VistA Imaging is in getting back the status from VistA as to the success of the image upload. The VA closely regulates the use of VistA Imaging at all hospital sites and will require the CIS to go through a certification process in order to be approved to use the VistA Imaging package. One of the criteria the CIS will be judged on is in providing feedback as to the successful upload of an image. The upload of images to VistA imaging is not instantaneous. It is a queued batch process where the time of completion depends on the current workload of the VistA Imaging processor. The VistA Gateway provides additional API’s to monitor the success of any upload request made to VistA Imaging. That status is tracked via the unique imaging identifier that was generated

as part of the image upload request.

There is a DICOM gateway available within the VistA Imaging system that CIS provider can subscribe to. This DICOM interface is outside of the scope of the VistA Gateway implementation. Those CIS vendor who wish to send DICOM images directly to VistA

Imaging should contact the DSS DataBridge Product Line Manager for further specifications.

Relevant Methods: Assuming an active connection is established, the VistA Gateway developer must call the “VI\_SendImageToVistaImaging” API. This API will either return a “0^Message” or a “Queue Number” to use as a tracking mechanism. After the image is queued for VistA Imaging filing the CIS is obligated to “track” the image. There is a tracking API that the VistA Gateway developer should use to understand the status of the image within VistA Imaging; refer to “VI\_GetImageStatus” API (providing the system has access to the VistA Imaging queue number). If the image is detected at any time to have a “Error” status, the CIS should invoke the “VI\_SendImageToVistaImaging” API again to re- queue.

Relevant Object Properties: The VistA Imaging import API is patient specific and TIU note specific, this means the VistA patient and TIU internal numbers (IFN) will need to be available to send the image to VistA Imaging. Normally the patient nu

mber will most likely be stored within the CIS database. If so, when images become available and a TIU note is associated with the desired image, the above mentioned “VI\_SendImageToVistaImaging” API method can be called. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of this document) should be called to set the object patient properties. The object stores numerous patient demographic data elements that can either be displayed to the user, create a new patient record, or update the HL7 ADT information stored in the offline SQL CIS database.

Interdependencies: A patient and TIU internal number must be available to process images into the VistA Imaging system. The image queue number is needed to acquire the details about the image status as it relates to the VistA Imaging system. A unique imaging identifier must be generated for each upload request to VistA Imaging and CANNOT BE REUSED for any other upload.

Best Practices: Processing images into the VistA Imaging system is normally invoked post- TUI note creation. The CIS should keep track of the requests to upload an image to VistA Imaging which should include the TIU note number, image, unique imaging identifier, and VistA Imaging status. The VistA Imaging status needs to be checked periodically until

either a “success” or “error” status is returned. The CIS should implement a way to re-send images that result in an “error” status. The CIS should also be aware sometimes the VistA Imaging processor can (on rare condition) fail to return a “success” status even though the upload of the image has completed and should therefore implement an administrative

process by which the status can be updated within the CIS database. Image(s) that are associated with the note should be collected by the CIS and placed into a publically available network share for the VistA Imaging system to consume. For those CIS images that are DICOM compliant an alternative HL7 based gateway is available; the CIS vendor must notify DSS if all (100%) of the CIS images are DICOM compliant. If so, an alterative solution is warranted and should be discussed.

VistA Gateway Radiology Ordering Information Implementation

The primary radiology data retrieval interface(s) is HL7 based. The radiology data will be sent to the CIS through an ORM HL7 outbound from VistA into the CIS. For more details on the HL7 DataBridge radiology interface, please be referred to the DataBridge HL7

Specification; particularly the ORM section.

As an alternative to the HL7 radiology event(s) DataBridge interfaces, the VistA Gateway has encapsulated much of the radiology results retrieval events through a variety of COM APIs. After the CIS system connects to VistA through the VistA Gateway, a variety of method calls are available to the VistA Gateway developers to process and display clinical information to the user. Two such method calls give the VistA Gateway developer the ability to retrieve a list of radiology orders from VistA as well as retrieving the details about a given radiology order. These APIs will allow the CIS to retrieve patient based radiology ACTIVE orders and their given details.

Relevant Methods: Assuming an active connection is established, the VistA Gateway developer must call the “CPRS\_GetPatientRadList” API. This API will either return a “-

1^Message” or a list of patient radiology orders and their associated internal numbers. This list view might be sufficient; if not, the user can drill down into the details of each radiology order list entry using the “CPRS\_GetPatientRadReport” API (providing the system has

access to the radiology order internal number).

Relevant Object Properties: The radiology APIs are patient specific, this means the VistA patient internal number (IFN) will need to be available to retrieve the radiology order(s) listing. Normally the patient number will most likely be stored within the CIS database. If so, when patient context changes within the CIS system, the patient number should be available to send into the above mentioned “CPRS\_GetPatientRadList” API method. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of

this document) should be called to set the object patient properties. The object stores numerous patient demographic data elements that can either be displayed to the user, create a new patient record, or update the HL7 ADT information stored in the offline SQL CIS database.

Interdependencies: A patient internal number must be available to acquire the patient radiology order(s) listing. The radiology order internal number is needed to acquire the details about the radiology order.

Best Practices: The radiology APIs have a master/detail relationship. Normally, the CIS system has pre-existing screens that display clinical master/detail information via a tree- view object that displays the master as a parent node and the details and children notes associated with the parent. Another way to display this information is to implement a list box that displays the radiology order listing and when a radiology order is selected an associated text box displays the details.

VistA Gateway Vitals Implementation

The primary vitals data retrieval interface(s) is HL7 based. The radiology data will be sent to the CIS through an OBX or OBR HL7 outbound from VistA into the CIS. For more details on the HL7 DataBridge radiology interface, please be referred to the DataBridge HL7 Specification; particularly the OBX or OBR section.

As an alternative to the HL7 vitals intake event(s) DataBridge interfaces, the VistA Gateway has encapsulated much of the vitals results retrieval events through a single COM API.

After the CIS system connects to VistA through the VistA Gateway, a method call is available to the VistA Gateway developers to process and display clinical information to the user. One such method call gives the VistA Gateway developer the ability to retrieve a list of the most current vitals that are on file for the given VistA patient in question.

Relevant Methods: Assuming an active connection is established, the VistA Gateway developer must call the “CPRS\_GetCurrentVitals” API. This API will either return a “-

1^Message” or a list of the current vitals values on file that that patient.

Relevant Object Properties: The vitals API is patient specific, this means the VistA

patient internal number (IFN) will need to be available to retrieve the vitals values.

Normally the patient number will most likely be stored within the CIS database. If so, when patient context changes within the CIS system, the patient number should be available to send into the above mentioned “CPRS\_GetCurrentVitals” API method. To keep the VistA Gateway object in sync with the system patient context, the “PAT\_SetNewPatient” method (see patient registration and patient selection sections of this document) should be called to set the object patient properties. The object stores numerous patient demographic data elements that can either be displayed to the user, create a new patient record, or update the HL7 ADT information stored in the offline SQL CIS database.

Interdependencies: A patient internal number must be available to acquire the patient vitals.

Best Practices: The vitals API deliver only the most current vital signs available for the patient in question. Normally, the CIS system should be collecting vitals values through the HL7 interface and storing these HL7 values within it’s CIS SQL database.

VistA Gateway Miscellaneous Supporting API Usage

There are miscellaneous “helper” APIs to assist in the VistA Gateway developer create the necessary API parameter values. The “MISC\_ConvertDisplayDateToVistADate” is an extremely valuable API; this API converts a window formatted date/time (mm/dd/yyyyy hh:mm.ss) to a VistA readable format. The “USR\_GetUserInformation” and “PAT\_GetPrimaryProvider” APIs allow the caller to retrieve VistA user details and the primary provider that is known to be assigned to the patient in question. The “MISC\_GetTableEntryData” API is another extremely useful method. This API allow a VistA Gateway developer to look up any field value within any VistA file (table) if the entry number is known i.e. patient race for patient “441” is ASIAN. At times, the CIS will need to “dump” legacy VistA data into it’s data repository to initialize it’s database tables to be in sync with VistA. There are a variety of PT\_\* API methods that allow the CIS vendor to acquire VistA table information to make their CIS database “VistA relevant”.

Glossary of Terms

Appointment: An outpatient encounter that has yet to occur.

COM: Component Object Model.

CPRS: VA Computerized Patient Record System. This system is a windows application, written in Delphi that allows a clinical to view clinical results and document orders and notes.

M: MUMPS programming language.

Nil: An empty string.

PCE: Patient Care Encounter system. This system is the module within VistA that stores all

CPT/ICD codes for all outpatient visits.

Pseudo-SSN: A unknown Social Security Number assigned to a VistA patient ending with

“P”.

RPC Broker: The client application and VistA server module the execute stored procedures within the VistA server database using TCP/IP communication.

RPC Broker Call: A stored procedure within the VistA database.

Sensitive Patient: A special category of a VistA patient filed within the VistA database. Examples of VistA sensitive patients are: employees, HIV/AIDS patients, an user accessing his/her own record, etc…

Stop Code: A categorical grouping of clinics normally by clinical treating specialties, i.e. Urology, Primary Care, etc…. A stop code is normally represented with a 3 digit whole number.

Surgical Case Number: A unique numerical identifier that represents an entry within the VistA surgical package. This number is also the internal entry number (IEN) for the surgical encounter in question.

TIU: Text Integration Utility. The electronic progress notes module within the Vista database.

UserClass: A VistA assigned user classification, i.e. Physician, Nurse, Coding Clerk. Used within the Patient Care Encounter system.

VA: The Department of Veterans’ Affairs.

VA FileManager: The VistA database explorer module.

VA FileManager Date/Time: A uniquely formatted string that is Y2K compliant. Formatting rules are as follows:

(Century-start with 1700) (Year)(Month)(Day)”.”(MilitaryTime)

3020415.1305\* (04/15/2002 01:05 PM)

3020415.1 (04/15/2002 10:00 AM)

2930415 (4/15/1993)

Visit: A checked-out appointment.

VistA: The Department of Veterans’ Affairs (VA) legacy database.

VistA Imaging: The VA clinical imaging system.

WideString: COM based string; maximum length: ~2^30 characters; memory required: 4 bytes to 2GB Unicode characters.

Appendix A -- Utility Functions (written in Delphi)

The “Piece” function. Many VistA return strings are formatted with certain delimiters. To assist in “tearing” the values apart the below function is provided. This function will return a sub string based on the delimiter and position the developer identifies.

S:=’1^A:B^3’;

{Piece will return “A:B” within var S1} S1:=Piece(S,’^’,2);

{Piece will return “A” within var S2} S1:=Piece(Piece(S,’^’,2),’:’,1);

function Piece(x: string; del: string; piece: integer) : string;

var

delIndex,pieceNumber,startScanIndex,firstCharIndex: integer;

begin startScanIndex:=1; pieceNumber :=0; delIndex :=1;

repeat

delIndex := Pos(del,Copy(x+del,startScanIndex,length(x)));

if delIndex > 0 then begin

inc(pieceNumber);

firstCharIndex := startScanIndex;

startScanIndex := startScanIndex + delIndex + Length(del) - 1;

end;

until (pieceNumber = piece) or (delIndex = 0);

if delIndex > 0 then

Result := Copy(x, firstCharIndex, delIndex-1)

else if (pieceNumber > 0) or (piece > 1) then

Result := '' else Result := x; end;

The “ConvertFileManDate” function. The VistA constructs unique date string internally within the database. To assist in the conversion of these strings to a windows date, the

below function is provided. This function will return a windows date to the caller.

var myDate:TDate

myDate:= ConvertFileManDate(‘3020301’);

{ConvertFileManDate will return a date of 03/01/2002}

function ConvertFileManDate(FMDateTime:String):TDateTime;

var

FirstYearString, YearString, MonthString, DayString: string;

begin

FirstYearString := IntToStr(StrToInt(Copy(FMDateTime, 1, 1)) + 17); YearString := Copy(FMDateTime, 2, 2);

MonthString := Copy(FMDateTime, 4, 2);

if (MonthString = '0') or (MonthString = '00') or (MonthString = '') then MonthString :=

'01';

DayString := Copy(FMDateTime, 6, 2);

if (DayString = '0') or (DayString = '00') or (DayString = '') then DayString := '01';

try

Result := EncodeDate(StrToInt(FirstYearString + YearString), StrToInt(MonthString), StrToInt(DayString));

except on exception do Result:=Now;

end;

end;

The “ConvertFileManTime” function. The VistA constructs unique time string internally within the database. To assist in the conversion of these strings to a windows time, the

below function is provided. This function will return a windows time to the caller.

var myTime:TTime

myDate:= ConvertFileManTime (‘141501’);

{ ConvertFileManTime will return a time of 02:15:01 PM}

function ConvertFileManTime(FMDateTime:String):TDateTime;

var sHour,sMin,sSec:String; Hour, Min, Sec: Word; begin

if Piece(FMDateTime,'.',2)<> '' then begin

if Piece(FMDateTime,'.',2)='24' then begin

Hour:=0; Min:=0; Sec:=0; end

else begin

sHour:=Copy(Piece(FMDateTime,'.',2), 1, 2); if Length(sHour)<2 then sHour:=sHour+'0'; sMin:=Copy(Piece(FMDateTime,'.',2), 3, 2); if Length(sMin)<2 then sMin:=sMin+'0'; sSec:=Copy(Piece(FMDateTime,'.',2), 5, 2); if Length(sSec)<2 then sSec:=sSec+'0';

if sHour<>'' then Hour:=StrToIntDef(sHour,0) else Hour:=0;

if sMin<>'' then Min:=StrToIntDef(sMin,0) else Min:=0; if sSec<>'' then Sec:=StrToIntDef(sSec,0) else Sec:=0; end;

end else begin

if FMDateTime='24' then begin

Hour:=0; Min:=0; Sec:=0; end

else begin

sHour:=Copy(FMDateTime, 1, 2);

if Length(sHour)<2 then sHour:=sHour+'0';

sMin:=Copy(FMDateTime, 3, 2);

if Length(sMin)<2 then sMin:=sMin+'0';

sSec:=Copy(FMDateTime, 5, 2);

if Length(sSec)<2 then sSec:=sSec+'0';

if sHour<>'' then Hour:=StrToIntDef(sHour,0) else Hour:=0;

if sMin<>'' then Min:=StrToIntDef(sMin,0) else Min:=0; if sSec<>'' then Sec:=StrToIntDef(sSec,0) else Sec:=0; end;

end;

try

Result:=EncodeTime(Hour, Min, Sec, 0); except on exception do Result:=Now; end;

end;

Appendix B – VistAGateway Additional Method/Property Request

Requested Method Description/Design: Related Existing Methods/Properties: Required Parameters:

Desired Return Results:

Please email to [Developers@documentstoragesystems.com](mailto:Developers@documentstoragesystems.com) or fax to 561-227-0208

Appendix C – CIS Partner Menu Option Table

Vendor Menu Option

Draeger DSIHA VISTA GATEWAY Docusys DSIHB VISTA GATEWAY Phillips DSIHC VISTA GATEWAY Picis DSIHD VISTA GATEWAY GE DSIHE VISTA GATEWAY

Attachment D – Remote Procedure Calls Used By The CIS Vendors

Broker Type Menu Option: DSIH\* VISTA GATEWAY

Call Name Associated API Method

DSIC DDR GETS ENTRY DATA MISC\_ConnectToVistA call

MISC\_GetTableEntryData call XWB GET VARIABLE VALUE MISC\_ConnectToVistA call DSIC DDR FILER MISC\_ConnectToVistA call DSIC DDR UPDATE FILE MISC\_ConnectToVistA call DSIC DDR FINDER MISC\_ConnectToVistA call TIU GET PERSONAL PREFERENCES MISC\_ConnectToVistA call VEJD XVA PERSON CLASS INFO MISC\_ConnectToVistA call

ORQQAL LIST CPRS\_GetPatientAllergyList call ORQQAL DETAIL CPRS\_GetAllergyDetails call VEJDSD GET PAT SCHED APPTS PCE\_ApptListByPatient call

VEJDSD GET SCHEDULED APPTS PCE\_GetApptsVisitsforStopCodeLocation call

VEJDSD GET STOP CODES APPT\_GetStopCodeListing call VEJDSD GET LOC BY STOP CODE APPT\_GetAllClinicsForAStopCode call VEJD PCE VISIT RETRIEVE PCE\_RetrieveVisitDetails call

VEJD PCE UPDATE PCE\_AddICDCodeToVisit call

DSIC PX PRIMARY PCE\_AddCPTCodeToVisit call PCE\_DeleteCPTEntryFromVisit call PCE\_DeleteICDEntryFromVisit call

ORQOR LIST CPRS\_GetPatientLabList call ORQOR DETAIL CPRS\_GetPatientLabDetail call ORWLR CUMULATIVE REPORT CPRS\_GetPatientLabCum call ORWLRR INTERIM CPRS\_GetPatientLabInterim call DG SENSITIVE RECORD ACCESS PAT\_SearchForVistaPatient call DG SENSITIVE RECORD ACCESS PAT\_SearchForVistaPatient call ORWPT ID INFO PAT\_SearchForVistaPatient call VEJD VESTING PAT\_SearchForVistaPatient call DG SENSITIVE RECORD BULLETIN PAT\_SearchForVistaPatient call ORQPT CLINIC PATIENTS PAT\_SearchForVistaPatient call VEJDDPT GET DEMO PAT\_SetNewPatient call

PAT\_GetDemographics call ORPRF HASFLG CPRS\_GetPatientFlagListing call ORPRF GETFLG CPRS\_GetPatientFlagReport call ORWORR AGET MED\_GetPatientMedList call ORWORR GET4LST MED\_GetPatientMedList call DSIC DDR LISTER MED\_GetPatientMedList call DSIC DATE CONVERT MED\_GetPatientMedList call

ORQQPL LIST CPRS\_GetPatientProblemList call ORQQPL DETAIL CPRS\_GetPatientProblemDetail call VEJDSD GET LOCATION MISC\_GetLocationInfo call

VEJD XUTIL DATE CONVERSION MISC\_ConvertDisplayDateToVistADate call

VEJDSD PRIMARY CARE PROV PAT\_GetPrimaryProvider call TIU CREATE RECORD TIU\_CreateTIURecord call TIU AUTHORIZATION TIU\_Authorization call

TIU UPDATE RECORD TIU\_EditTIUUnSignedNote call

TIU LOAD RECORD FOR EDIT TIU\_RetrieveUnsignedNoteText call

TIU GET PERSONAL PREFERENCES USR\_GetUserLocation call DSIC DDR GETS ENTRY DATA USR\_GetUserInformation call TIU LONG LIST OF TITLES TIU\_ShowTitleSearchDialog call

ORWU VALIDSIG CPRS\_ValidateESignature call

TIU SIGN RECORD TIU\_SignTIURecord call

MAG4 REMOTE IMPORT VI\_SendImageToVistaImaging call DSIV DM GET STATUS VI\_GetImageToVistaImaging call ORWRA IMAGING EXAMS1 CPRS\_GetPatientRadList call ORWRP REPORT TEXT CPRS\_GetPatientRadReport call ORQQVI VITALS CPRS\_GetCurrentVitals call

VEJD XUTIL DATE CONVERSION MISC\_ConvertDisplayDateToVistADate call

VEJD REPORTS PT\_\* calls

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Change Description |
| 1.0.1 | 03/26/2008 | J Flejter | Initial Definition |
| 1.0.2 | 03/28/2008 | J Flejter | Integrated V Hornback VistA Imaging Information |
| 1.0.3 | 03/28/2008 | J Flejter | Identify Remote Procedure Calls within Appendix D |