RAPTOR Data Layer Functionality

Contents

[Fundamental Data Concepts and Terms in RAPTOR](#h.gjdgxs)

[Unique Tracking ID](#h.30j0zll)

[Site ID](#h.1fob9te)

[Application Data Layer](#h.3znysh7)

[Ticket Workflow State Codes](#h.2et92p0)

[Worklist Modes](#h.3dy6vkm)

[Ticket Tracking](#h.4d34og8)

[Class TicketTrackingData](#h.2s8eyo1)

[Constructor](#h.17dp8vu)

[POC Background](#h.3rdcrjn)

[Function getTicketWorkflowState($sTrackingID)](#h.26in1rg)

[Function markTicketLocked($sTrackingID, $sUserID, $sStateCode)](#h.lnxbz9)

[Function markTicketUnlocked($sTrackingID)](#h.35nkun2)

[Function setTicketWorkflowState($sTrackingID, $sStateCode)](#h.1ksv4uv)

[Worklist Data](#h.44sinio)

[Class WorklistData](#h.2jxsxqh)

[Constructor](#h.z337ya)

[POC Background](#h.3j2qqm3)

[Function getWorklistModeName($sWMODE)](#h.1y810tw)

[Function getWorklistRows($oContext)](#h.4i7ojhp)

[Function getOneWorklistRow($sTrackingID)](#h.3whwml4)

[Protocol Data](#h.2bn6wsx)

[POC Background](#h.3as4poj)

[Class DashboardData](#h.1pxezwc)

[Constructor](#h.2p2csry)

[Function getDashboardDetails($oContext)](#h.147n2zr)

[Dashboard Associative Array Labels](#h.23ckvvd)

[Class ProtocolSupportingData](#h.32hioqz)

[Constructor](#h.1hmsyys)

[Function getOrderOverview()](#h.41mghml)

[Function getMedicationsDetail()](#h.2grqrue)

[Function getVitalsSummary()](#h.vx1227)

[Function getVitalsDetail()](#h.3fwokq0)

[Function getAllergiesDetail()](#h.1v1yuxt)

[Function getProcedureLabsDetail()](#h.4f1mdlm)

[Function getDiagnosticLabsDetail()](#h.2u6wntf)

[Function getDoseHxDetail()](#h.19c6y18)

[Function getPathologyReportsDetail()](#h.3tbugp1)

[Function getSurgeryReportsDetail()](#h.28h4qwu)

[Function getProblemsListDetail()](#h.nmf14n)

[Function getNotesDetail()](#h.37m2jsg)

[Function getRadiologyReportsDetail()](#h.1mrcu09)

[Miscellaneous Helper Classes](#h.46r0co2)

[Class UserInfo](#h.2lwamvv)

[Functions to Read Data for ONE USER](#h.111kx3o)

[Functions to Read Data for ALL USERS](#h.3l18frh)

[Functions to Change Persistent Data](#h.206ipza)

[Class ScheduleInfo](#h.4k668n3)

[Constructor](#h.2zbgiuw)

[Functions to Read Data](#h.1egqt2p)

[Functions to Change Persistent Data](#h.3ygebqi)

[Class ImageInfo](#h.2dlolyb)

[Constructor](#h.sqyw64)

[Functions to Read Data](#h.3cqmetx)

[Class GraphData](#h.1rvwp1q)

[Constructor](#h.4bvk7pj)

[Functions to Get Values](#h.2r0uhxc)

[Class ProtocolLibraryInfo](#h.1664s55)

[Constructor](#h.3q5sasy)

[Functions to Read Data](#h.25b2l0r)

[Class Context](#h.kgcv8k)

[Functions for Reading Data](#h.34g0dwd)

[Functions for Authentication with VISTA Subsystem](#h.1jlao46)

[Class PatientCare](#h.43ky6rz)

[Constructor](#h.2iq8gzs)

[Functions to Interact with Protocol Values](#h.xvir7l)

[Functions to Interact with Exam Values](#h.3hv69ve)

[Functions to Interact with QA Values](#h.1x0gk37)

[Class DiagnosticCheck](#h.4h042r0)

[Functions to Check Status](#h.2w5ecyt)

[Class ListOptions](#h.1baon6m)

[Functions to Read Data](#h.3vac5uf)

[Class BoilerplateTextHelpers](#h.2afmg28)

[Functions to Read Data](#h.pkwqa1)

[Global Variables](#h.39kk8xu)

[Local RAPTOR Database](#h.48pi1tg)

[Revision History](#h.2nusc19)

# Fundamental Data Concepts and Terms in RAPTOR

RAPTOR is primarily a tool for protocoling imaging studies. The following terms are sometimes used interchangeably in this context: study, order, procedure. For purposes of this document, we will use the term “ticket” to describe a study/order/procedure that is being handled by RAPTOR.

## Unique Tracking ID

Each ticket in RAPTOR has to have a unique “Tracking ID” (aka, “Ticket ID” or “Ticket Number”). This tracking id is assigned within RAPTOR so that no two study/order/procedure (“ticket”) ever have the same number and the number always maps back to the original study/order/procedure (“ticket”).

The ticket tracking number is used within RAPTOR to associate protocol and workflow details with the original study/order/procedure.

*NOTE: The RAPTOR Ticket ID is a dash delimited concatenation of site identifier and IEN value returned by MDWS. Format of the RAPTOR Tracking ID is <siteid>-<IEN>.*

### Site ID

Each site has a unique ID that is configured for the RAPTOR installation of that site.

## Application Data Layer

The RAPTOR application interacts with all external systems and its local database via a data layer of custom PHP classes and functions. Those classes and functions are described in this document. All classes are expected to be part of the “**raptor**” namespace.

*NOTE: If a function returns a single dimension array of values, that array should be an associative array unless otherwise stated. This will ensure a greater level of assurance that the values are being used for their intended purpose in the consuming modules.*

### Ticket Workflow State Codes

A ticket can only be in one of the following RAPTOR states. The data layer is expected to work with these codes for filtering purposes.

*Table 1 - RAPTOR Ticket States*

|  |  |
| --- | --- |
| State Name | State Code |
| Active | AC |
| Approved | AP |
| Collaborative | CO |
| Review | RV |
| Protocol Acknowledged | PA |
| Inactive | IA |
| Exam Complete | EC |
| QA Complete | QA |

### Worklist Modes

One of the worklist filtering criteria is via the worklist mode. The mode filtering works as described in Table 2. The data layer is expected to work with these codes and support this filtering.

*Table 2 - Worklist Modes*

|  |  |  |
| --- | --- | --- |
| Mode Name | Code | Description |
| Protocol | P | Display any tickets that are in any of the following workflow states:   * Active * Collaborative * Review |
| Examination | E | Display any tickets that are in any of the following workflow states:   * Approved * Protocol Acknowledged |
| Interpretation | I | Display any tickets that are in any of the following workflow states:   * Exam Complete |
| QA | Q | Display any tickets that are in any of the following workflow states:   * Exam Complete * Inactive * QA Complete |

# Ticket Tracking

A key requirement of the data layer in RAPTOR is that it can associate RAPTOR data with externally sourced data. Internally each ticket gets a unique tracking ID for that purpose.

## Class TicketTrackingData

This is a utility class that does not map to any particular page and should be coded into a file called **data\_ticket\_tracking.php**.

### Constructor

No constructor is necessary for this class. Methods will have static behavior.

### POC Background

The implementation of the tracking identifier in the POC was not sufficient for production purposes. The concept was that there are a set of values that exist in an externally sourced study/order/procedure that can be consistently converted into the same tracking number each time those inputs are provided.

### Function getTicketWorkflowState($sTrackingID)

Returns one of the values from Table 1. If no state has been recorded for the ticket, then this function returns the status code for the “Active” state.

*This function will be implemented by the presentation layer team.*

### Function markTicketLocked($sTrackingID, $sUserID, $sStateCode)

Calling this function marks a ticket as being edited by a RAPTOR user. The $sStateCode is optional and if not NULL then the ticket is placed into the indicated state (see Table 1 for values).

*This function will be implemented by the presentation layer team.*

### Function markTicketUnlocked($sTrackingID)

Calling this function clears any lock that might have associated the ticket as being edited by a RAPTOR user.

*This function will be implemented by the presentation layer team.*

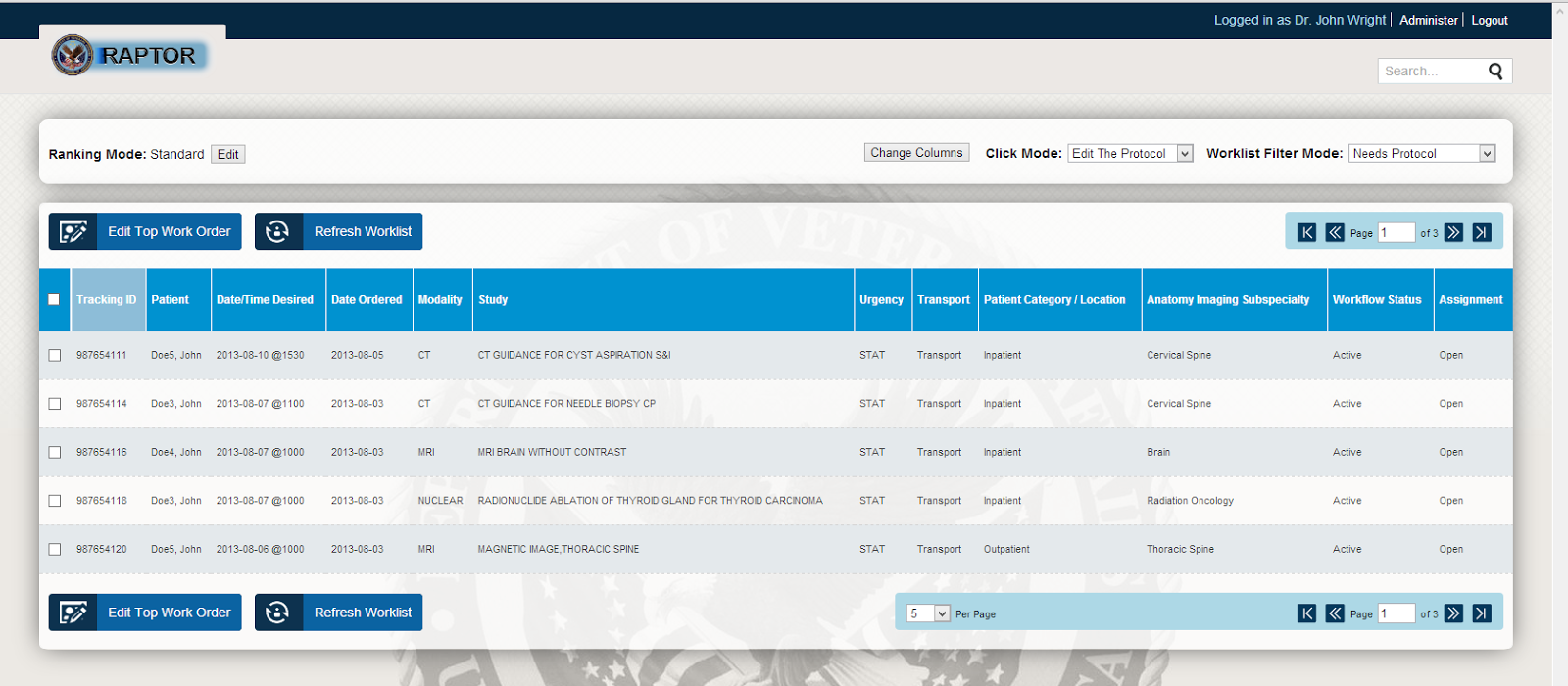
### Function setTicketWorkflowState($sTrackingID, $sStateCode)

Calling this function sets a ticket as being in a particular RAPTOR workflow state (see Table 1 for values).

*This function will be implemented by the presentation layer team.*

# Worklist Data

The worklist shows all the available orders for a user to protocol. The orders might be in any state. There is also a request from the VA that they be able to create orders in RAPTOR for protocol purposes.



*Figure 1 - Sample Worklist Page*

The following classes and functions provide the functionality for the application to display and for the user to interact with the worklist data.

## Class WorklistData

This class returns worklist specific data and should be coded into a file called **data\_worklist.php**.

### Constructor

Takes a Context instance as input. The Context is a class managed by the RAPTOR application and contains key values such as UserID of the current user and various filter and sorting options they have selected.

### POC Background

You will find relevant data implementation details in the old **raptor\_worklist.php** and **worklist.php** files. The primary presentation layer function is **\_raptor\_blocks\_get\_worklist\_html** found in the old **page\_content\_html.inc** file.

### Function getWorklistModeName($sWMODE)

This is a static function that returns the word associated with the code.

/\*\*

\* Convert the code into a word

\* @param type $sWMODE

\* @return string the word associated with the code

\*/

static function getWorklistModeName($sWMODE)

{

if($sWMODE=='P'){

$sName="Protocol";

} elseif ($sWMODE=='E'){

$sName="Examination";

} elseif ($sWMODE=='I'){

$sName="Interpretation";

} elseif ($sWMODE=='Q'){

$sName="QA";

} else {

die("Invalid WorklistMode='$sWMODE'!!!");

}

return $sName;

}

### Function getWorklistRows($oContext)

Return a multi-part hybrid associative array of all worklist tuples satisfying the filter criteria implied by the instance of Context class passed in as a parameter.

*Table 3 - Labeled Items of the WorklistRows Array*

|  |  |  |
| --- | --- | --- |
| Item | Label | Value Description |
| 1 | Pages | Number of pages found |
| 2 | Page | Current page for which rows are being returned |
| 3 | RowsPerPage | Rows appearing on each page |
| 4 | DataRows | An array of non-associative array data rows. |

Each array data row contains the values shown in Table 4.

*Table 4 – Values in the Worklist DataRows Array*

|  |  |  |
| --- | --- | --- |
| Array Offset | Name | Description |
| 0 | Tracking ID | This is a unique ID assigned by the RAPTOR application to the order. At any time, the raptor application can request this order simply by providing this value as a parameter to the data layer functions. |
| 1 | Patient ID | The patient ID value in MDWS. (Also known as the ‘DFN’ value.) |
| 2 | Patient Name | “Last name, First name” of the patient |
| 3 | Date/Time Desired | The date and time that the procedure is requested to occur |
| 4 | Date Ordered | The date (and time) that the order was placed |
| 5 | Modality | The modality of the procedure |
| 6 | Study | Description of the ordered procedure |
| 7 | Urgency | One of several values: STAT, URGENT, *blank* |
| 8 | Transport | One of several keywords |
| 9 | Patient Category / Location | One of several keywords |
| 10 | Anatomy Image Subspecialty | One or more anatomy keywords (As of 6/11/2014 this value is still not being returned by data layer.) |
| 11 | Workflow Status | The current RAPTOR workflow state of this ticket  (See join with **raptor\_ticket\_tracking** table. Show the **workflow\_state** field value.) |
| 12 | Assignment | Null if no assignment, else associative array with following keys:   * uid ← System user ID value (number) * username ← System login name * fullname ← Actual name of the user * requester\_notes\_tx ← Notes for the user * requested\_dt ← When requested |
| 13 | Order Status | The status keyword from external system (same value as seen in CPRS) |
| 14 | Editing User ID | If the ticket is currently being edited by a RAPTOR user, this is the User ID of that user. Otherwise this is blank. |
| 15 | ScheduleInfo | Associative array of schedule information.   * EventDT – Datetime instance for which the event is scheduled. If not scheduled, this is not set. * LocationTx – Room number if set. * ConfirmedDT – Blank if not confirmed, else this is a datetime instance. * CanceledDT – Blank if not canceled, else this is a datetime instance. * ShowTx – Suggested text to show in worklist. (E.g., “TBD” if not set, else “MMMMDDYY@HH:MM”)   (See join with **raptor\_schedule\_track** table.) |
| 16 | CPRS Code | “Field 9” from CPRS (this is a 5 digit number such as 73615 etc.)  (As of 6/11/2014 this value is still not being returned by data layer.) |
| 17 | Imaging Type | “Field 12” from CPRS. This is a text value. Example values are as follows…   * ANGIO/NEURO/INTERVENTIONAL * CT SCAN * GENERAL RADIOLOGY * MAGNETIC RESONANCE IMAGING * MAMMOGRAPHY * NUCLEAR MEDICINE |
| 18 | Ranking Score | An array with following elements   * offset0 -> numeric value representing the ranking of this row for the currently logged in user. * offset1-> array of scoring breakdown |

The presentation layer sorts and filters the worklist array such that only a subset of the rows and columns are displayed.

**Filtering Note:** If an order/procedure/study is ACTIVE or PENDING in CPRS and there is no existing Tracking ID already associated with it in RAPTOR, then for RAPTOR purposes that order/study/procedure is assumed to be in the RAPTOR “Active” state.

### Function getOneWorklistRow($sTrackingID)

Return all the values for only one worklist tuple as described in Table 4. The $sTrackingID uniquely identifies the item.

# Protocol Data

The protocol data context for RAPTOR has a main page in which the user enters/edits protocol information and several sub-tab pages in which they primarily view supporting detail information.

*Table 5 - Tabs of the Protocol Page*

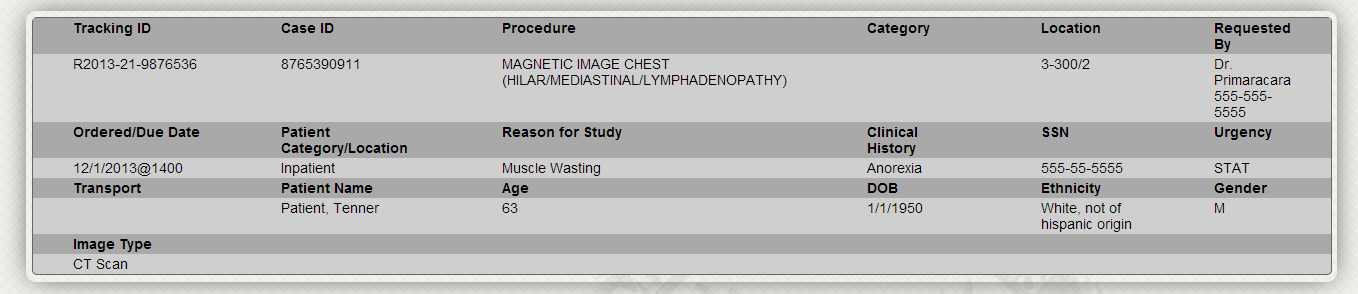
|  |  |  |
| --- | --- | --- |
| Order | Tab Name | Description |
| 1 | Protocol | This is the default selected tab when a user lands on the Protocol Page. This tab area is where all the protocol settings are captured and some read-only background information is made available to help with the protocol process. This is also the tab area where a technician enters notes during the procedure. |
| 2 | Medications | Displays medication details |
| 3 | Vitals | Displays vitals details |
| 4 | Allergies | Displays allergy details |
| 5 | Labs | Displays lab details depending on the type of ticket |
| 6 | Dose Hx | Displays dose details |
| 7 | Clinical Reports | Displays clinical report details |
| 8 | Problems List | Displays problem details |
| 9 | Notes | Displays note details |
| 10 | Radiology Reports | Displays radiology report details |
| 11 | *Protocol Library* | *This is where we find a listing of all the existing protocols available at the hospital center* |

## POC Background

The content for the pages in the POC are primarily produced by the **\_raptor\_blocks\_get\_detail\_html** function of the old **page\_content\_html.inc** file. You can navigate back from that point to see how the data is extracted from MDWS in the POC.

## Class DashboardData

This class provides the dashboard (formerly known as “boilerplate” in some documents) area content data. This data is a presentation of key information specific to the selected ticket. The look and feel of the dashboard is illustrated in Figure 2. The presentation layer handles all the formatting of the data.



*Figure 2 - Look and feel of Dashboard Area (Specific layout subject to revision)*

The implementation of this class is found in the **data\_dashboard.php** file.

### Constructor

This class needs no constructor because the methods have static behavior.

### Function getDashboardDetails($oContext)

This function returns one tuple with the values described in Table 6. The tuple should be an instance of an associative array with each value label reasonably matching the label name show in Table 6.

*Table 6 – Dashboard Content*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Dashboard Label | Expected Max Characters | Description |
| 1 | Tracking ID | 50 | RAPTOR unique tracking ID for the procedure. RAPTOR must create this number because it stores data locally. |
| 2 | Case ID | 50 | From VistA Case Name field |
| 3 | Procedure | 250 | From VistA Rad Procedure field. |
| 4 | Category | 50 | From VistA Category field |
| 5 | Location | 50 | Room/Bed for inpatients |
| 6 | Requested By | 100 | Name and Phone# of requesting physician |
| 7 | Requested Date | 10 | The MM/DD/YYYY date on which the order was placed. |
| 8 | Scheduled Date | 10 | The MM/DD/YYYY date on which the procedure is scheduled to take place |
| 9 | Patient Category | 12 | ???? inpatient vs outpatient ??? |
| 10 | Reason for Study | 250 | From VistA Reason field |
| 11 | Clinical History | 250 | From VistA Clinical History field |
| 12 | SSN | 11 | From VistA Patient SSN field (###-##-####) |
| 13 | Urgency | 20 | From VistA Urgency field |
| 14 | Transport | 50 | From VistA Transport field |
| 15 | Patient Name | 80 | From VistA Patient Name field. Display as ***Last, First***. |
| 16 | Age | 3 | From VistA Age field |
| 17 | DOB | 10 | From VistA DOB field. Display as MM/DD/YYYY |
| 18 | Ethnicity | 10 | From VistA Ethnicity field. |
| 19 | Gender | 1 | From VistA Gender field. Display as M or F. |
| 20 | Image Type | 20 | From VistA Image Type field (CPRS field 12) |
| 21 | CPRS Code | 5 | From CPRS field 9 |
| 22 | NA | ? | From MDWS call as mpiPid |
| 23 | NA | ? | From MDWS call as mpiChecksum |
| 24 | NA | ? | From MDWS patient ID |

### Dashboard Associative Array Labels

The instance of the dashboard is returned as an associative array. The following snippet illustrates the syntax for an associate array that contains only 2 of the required values.

*return Array(“Tracking ID”=>$sTrackingID, “CaseID” => $sCaseID);*

The expected labels for the associative array are described in Table 7.

*Table 7 - Dashboard Associative Array Labels*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Dashboard Label | Array Key | Value Description |
| 1 | Tracking ID | Tracking ID | Contains string value |
| 2 | Case ID | CaseID | Contains string value |
| 3 | Procedure | Procedure | Contains string value |
| 4 | Category | Category | Contains string value |
| 5 | Location | PatientLocation | Contains string value |
| 6 | Requested By | RequestedBy | Contains string value |
| 7 | Requested Date | RequestedDate | Contains string value **MM/DD/YYYY** |
| 8 | Scheduled Date | ScheduledDate | Contains string value **MM/DD/YYYY @ HH:MM** |
| 9 | Patient Category | PatientCategory | Contains string value |
| 10 | Reason for Study | ReasonForStudy | Contains string value |
| 11 | Clinical History | ClinicalHistory | Contains string value |
| 12 | SSN | PatientSSN | Contains string value **###-##-####** |
| 13 | Urgency | Urgency | Contains string value |
| 14 | Transport | Transport | Contains string value |
| 15 | Patient Name | PatientName | Contains string value **Last, First** |
| 16 | Age | PatientAge | Contains string value |
| 17 | DOB | PatientDOB | Contains string value **MM/DD/YYYY** |
| 18 | Ethnicity | PatientEthnicity | Contains string value |
| 19 | Gender | PatientGender | Contains string value |
| 20 | Image Type | ImageType | Contains string value |
| 21 | CPRS Code | CPRSCode | Contains numberic value ##### |
| 22 | NA | mpiPid | This is the patient ICN. This needed for us to interface with RAPTOR VIX. |
| 23 | NA | mpiChecksum | The checksum for patient ICN |
| 24 | NA | PatientID | The patient ID value in MDWS. (Also known as the ‘DFN’ value.) |

## Class ProtocolSupportingData

This class provides the data that is displayed in the static information area of the main protocol tab and also the detailed content for the specialized tabs.

### Constructor

This constructor should take an instance of the ***Context*** class.

### Function getOrderOverview()

Return a single associative array with the following values for the ticket.

1. Requested By
2. Primary Care Physician
3. Attending Physician
4. Requested Study
5. Reason for Study

### Function getMedicationsDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. Medication
2. At Risk
3. Status

### Function getVitalsSummary()

Return an associative array of all data to display. There is only one row of key value pairs and the keys are the labels meant for display to the user.

1. Temperature
2. Heart Rate
3. Blood Pressure
4. Height
5. Weight
6. Body Mass Index

Each of the keys above is associated with an array with the following content.

1. Date of Measurement
2. Measurement Value

Programmer Note: Do not look for the keys listed here. Instead simply use them as labels. Additional keys and their values may be added in the future and we would like them to simply flow through to the display.

### Function getVitalsDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. Date Taken
2. Temp
3. Height
4. Weight
5. BMI
6. Blood Pressure
7. Pulse
8. Resp
9. Pain
10. C/G
11. Pox
12. CVP
13. Blood Glucose

### Function getAllergiesDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. DateReported
2. Item
3. CausativeAgent
4. SignsSymptoms ← array(‘Snippet’ => *short text*, ‘Details’ => *full text*)
5. DrugClasses ← array(‘Snippet’ => *short text*, ‘Details’ => *full text*)
6. Originator
7. ObservedHistorical ← array(‘Snippet’ => *short text*, ‘Details’ => *full text*)

### Function getProcedureLabsDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. Date
2. PLT
3. PT
4. INR
5. PTT
6. HCT
7. Ref

### Function getDiagnosticLabsDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. DiagDate
2. Creatinine
3. eGFR
4. Ref

### Function getDoseHxDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. Action
2. Exam Date
3. Procedure Type
4. Height
5. Weight
6. BMI
7. CTDIvol (mGy)
8. DLP (mGy cm)
9. Radiotracer Dose (mCi)
10. Dose Type
11. Entry State

### Function getPathologyReportsDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. Title
2. ReportDate
3. Details  Array of details
4. Snippet  Short text suitable for a table row entry

### Function getSurgeryReportsDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. Title
2. ReportDate
3. Details  Array of details
4. Snippet  Short text suitable for a table row entry

### Function getProblemsListDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. Title
2. Onset Date
3. Details  Array of details
4. Snippet  Short text suitable for a table row entry

### Function getNotesDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. Type
2. Date
3. Details  Array of details
4. Snippet  Short text suitable for a table row entry

### Function getRadiologyReportsDetail()

Return an associative array of all data to display. Each row of array should contain the following data:

1. Title
2. Date
3. Details ← Array of details
4. Snippet ← Short text suitable for a table row entry
5. AccessionNumber ← Value from [accessionNumber]
6. CaseNumber ← Value from [caseNumber]
7. ReportID ← Value from [id]

# Miscellaneous Helper Classes

These are classes that do not relate to one specific ticket.

## Class UserInfo

This class provides information about a user. The constructor takes a UID (user id) as the only input. This class is implemented in the **data\_user.php** file.

### Functions to Read Data for ONE USER

* Function getUserID()
  + Returns the RAPTOR User ID of the currently logged in user.
* Function getUserName()
  + Returns the system **username** of the currently logged in user.
* Function getUserFirstName()
  + Returns the First Name of the currently logged in user.
* Function getUserLastName()
  + Returns the Last Name of the currently logged in user.
* Function isSiteAdministrator()
  + Returns TRUE if user is a site administrator, else FALSE.
  + This is same as checking for role\_nm == ‘Site Administrator’.
* Function isEnabled()
  + Returns TRUE if user is enabled, else returns FALSE.
  + This result is same as value returned for “AA1” key of getSystemPrivileges() result.
* Function getRoleName()
  + Returns the role name for the user
* Function getUserNameTitle
  + Return the title, if any. (e.g., Dr, etc)
* Function getUserNameSuffix
  + Return the suffix if any. (e.g., PhD, etc)
* Function getPreferredEmailAddress()
  + Return the formatted string
* Function getPreferredPhoneNumber()
  + Return the formatted string
* Function getGroupMemberships()
  + Return array of codes
* Function getModalityPreferences()
  + Return array of modalities preferred by this user.
* Function getServicePreferences()
  + Return array of services preferred by this user.
* Function getWeightedAnatomyPreferences()
  + Return array of arrays. Each array contains keywords, first array has more weight than the next, etc.
* Function getSpecialistModality()
  + Return array of arrays modality abbreviations in which the user specializes.
* Function getSpecialistService()
  + Return array of services in which the user specializes.
* Function geWeightedtSpecialistAnatomy ()
  + Return array of arrays. Each array contains keywords, first array has more weight than the next, etc.
* Function getSystemPrivileges()
  + Return an associative array with key is the ID and value is TRUE or FALSE.
  + See the RAPTOR User Management Tech Guide for the complete list of codes and an explanation for their purpose.
* Function getWorklistColumnCustomization()
  + Return value from **raptor\_user\_profile.worklist\_cols** field.

### Functions to Read Data for ALL USERS

* Function getAllUsers($bEnabled)
  + Return an array of UserInfo instances for all users of the RAPTOR system. If $bEnabled is set, then only return the enabled users when that value is TRUE, else return the disabled users if that value is FALSE.

### Functions to Change Persistent Data

* Function createUser($aUserInfo)
  + Pass an associative array of user information to create a new user record.
  + Returns UID value of new user record if success, else associative array containing the error details.
    - ERRNUMBER
    - ERRDETAIL
* Function updateUser($aUserInfo)
  + Pass an associative array of user information to create a new user record.
    - The UID must be passed in as key “UID” in the associative array.
    - The UAN1 cannot be changed.
  + Returns TRUE if success, else associative array containing the error details.
    - ERRNUMBER
    - ERRDETAIL

## Class ScheduleInfo

This class provides information about a scheduled event. This class is implemented in the **data\_schedule.php** file.

### Constructor

Pass in the **TrackingID**

### Functions to Read Data

* Function getDate()
  + Returns null if not scheduled, else returns date.
* Function getStartTime()
  + Returns null if not scheduled, else returns start time.
* Function getDuration()
  + Returns null if not scheduled, else returns duration in minutes as an integer.
* Function getEndTime()
  + Returns null if not scheduled, else returns end time (as computed by applying duration to the start time.)
* Function getLocation()
  + Returns null if not scheduled, else returns location string.
* Function getNotes()
  + Returns null if not scheduled, else returns the notes.
* Function getLastUpdated()
  + Returns null if not scheduled, else returns associative array with DATETIME key (with date and time of the last update) and UID (with value of user that last updated the record) key.

### Functions to Change Persistent Data

* Function createRecord($aScheduleInfo)
  + Pass an associative array of schedule information to create a new user record.
  + Returns TRUE if success, else returns an associative array containing the error details.
    - ERRNUMBER
    - ERRDETAIL
* Function updateRecord($aUserInfo)
  + Pass an associative array of schedule information to create a new user record.
    - The TrackingID must be passed in as key “TrackingID” in the associative array.
  + Returns TRUE if success, else associative array containing the error details.
    - ERRNUMBER
    - ERRDETAIL
* Function deleteRecord($sTrackingID)
  + Pass an associative the tracking id of the scheduled event to delete.
  + Returns TRUE if success, else associative array containing the error details.
    - ERRNUMBER
    - ERRDETAIL

## Class ImageInfo

This class provides information about images. The constructor takes a UID (user id) as the only input. This class is implemented in the **data\_images.php** file.

### Constructor

None.

### Functions to Read Data

* Function getImageURLS($sTrackingID)
  + Returns array of associative arrays. Each array contains following keys:
    - THUMBNAILURL
    - Others TBD
* Function getImageViewerURL($sTrackingID)
  + Returns the URL to the VIX image viewer for a specific ticket of images.

## Class GraphData

This class provides data points for graphs. The constructor takes a UID (user id) as the only input. This class is implemented in the **data\_graphdata.php** file.

### Constructor

Pass in the **TrackingID**

### Functions to Get Values

* Function getThumbnailGraphValues()
  + Returns associative array of values for the thumbnail chart on the protocol page.
    - Associative array has no more than most recent 10 temperature values of the patient.
      * Each item in array is an associative array with following keys value pairs
        + date
        + temperature
* Function getVitalsGraphValues()
  + Returns associative array of values for the vitals chart on the vitals tab.
    - Associative array has no more than most recent 10 temperature and pulse values of the patient for those same dates.
      * Each item in array is an associative array with following keys value pairs
        + date
        + temperature
        + tempFlag  Values are **MISSING** and **NOCIRCLE**
        + pulse
        + pulseFlag  Values are **MISSING** and **NOCIRCLE**
      * If the flag value is **MISSING**, then an empty circle will be displayed in the graph to indicate we are plotting an interpolated point. Use this when you have a temperature but do not have a pulse for a date, or vice versa for the item you did not have in that case.
      * If the flag value is **NOCIRCLE** then the point is plotted with no special indicator at that spot.
* Function getLabsGraphValues()
  + Returns associative array of values for the labs chart on the protocol page.
    - Associative array has no more than most recent 10 eGFR values of the patient.
      * Each item in array is an associative array with following keys value pairs
        + date
        + egfr

## Class ProtocolLibraryInfo

This class provides information about protocol library. This class is implemented in the **data\_protocollibrary.php** file.

### Constructor

None.

### Functions to Read Data

* Function getProtocols($bActive)
  + Returns array of associative arrays, one array for each protocol in the library. Each array contains following keys:
    - ProtocolID
    - UploadedDate
    - ActiveStatus
    - VersionNumber
    - Modality
    - Service
    - Hydration
    - Radioisotope
    - Contrast
    - Allergy
    - Claustrophobic
    - Sedation
    - ConsentRequired
    - ProtocolNotes
  + If $bActive parameter is set, then only return active protocols for value TRUE, in-active protocols for value FALSE.
* Function getProtocolJSON($sProtocolID)
  + Returns the JSON string of information for the selected protocol. (See the ***RAPTOR Protocol Library Tech Guide*** for details on the JSON structure.)

## Class Context

This class is a singleton instance for the application session and describes key information for the user’s session. This class is implemented in the **data\_context.php** file.

### Functions for Reading Data

* Function static getSiteID()
  + Returns the three digit site ID of the installation.
* Function getUserInfo()
  + Returns an instance of the UserInfo class for the currently logged in RAPTOR user.
* Function getWorklistMode()
  + Returns the currently selected worklist mode for the session. The value is a code as shown in Table 2.
* Function getSelectedTrackingID()
  + Returns the tracking ID of the currently selected ticket or NULL if none is selected

### Functions for Authentication with VISTA Subsystem

The following helper functions of the Context class are the API for RAPTOR to integrate with the Vista authentication systems.

* Function authenticateSubsystems($sVistaUserID, $sVAPassword)
  + Utilizes MDWS for authentication.
  + Returns empty string if authenticated OK, else associative array with following keys: ERRNUM, ERRSUMMARY, ERRDETAIL
* Function isAuthenticatedInSubsystem()
  + Utilizes MDWS to see if user is already authenticated
  + Returns FALSE if not authenticated with subsystem.
  + Returns TRUE if currently authenticated in subsystem.
* Function logoutSubsystems()
  + Utilizes MDWS to logout.
  + Returns empty string if logout OK, else associative array with following keys: ERRNUM, ERRSUMMARY, ERRDETAIL

## Class PatientCare

The methods of this class enable update of protocol, exam, and qa data of a ticket. This class is implemented in the **data\_patientcare.php** file.

### Constructor

The constructor takes the **TrackingID** of the ticket as the only parameter.

### Functions to Interact with Protocol Values

* Function getProtocol()
  + Returns associative array of all the protocol values.
* Function writeProtocol($aProtocolInfo)
  + Pass in an associative array containing all the values to write.

### Functions to Interact with Exam Values

* Function getExam()
  + Returns associative array of all the exam values.
* Function writeExam($aProtocolInfo)
  + Pass in an associative array containing all the values to write.

### Functions to Interact with QA Values

* Function getQA()
  + Returns associative array of all the QA values.
* Function writeQA($aProtocolInfo)
  + Pass in an associative array containing all the values to write.

## Class DiagnosticCheck

This class provides information about operation status of data appliances/services. This class is implemented in the **data\_diagnosticcheck.php** file.

### Functions to Check Status

* Function getMySQLStatus()
  + Returns associative array with following keys.
    - Status  Values are OK and ERROR
    - Code  Values are 0 for OK, else an error code number
    - Message  Full error message if error, else empty.
* Function getMDWSStatus()
  + Returns associative array with following keys.
    - Status  Values are OK and ERROR
    - Code  Values are 0 for OK, else an error code number
    - Message  Full error message if error, else empty.
* Function getVIXStatus()
  + Returns associative array with following keys.
    - Status  Values are OK and ERROR
    - Code  Values are 0 for OK, else an error code number
    - Message  Full error message if error, else empty.

## Class ListOptions

This class provides values for dropdown lists. It is implemented in the **data\_listoptions.php** file.

### Functions to Read Data

* Function getHydrationOptions($type, $modality)
  + Pulls values from **raptor\_list\_hydration** table.
  + Values for $type are Oral or IV to filter on **type\_nm** field value
  + Values for $modality are CT, MR, and NM
    - CT returns rows where ct\_yn=1
    - MR returns rows where mr\_yn=1
    - NM returns rows where nm\_yn=1
  + Returns simple array of values in **option\_tx** field.
* Function getRadiologicalPharmaOptions($type, $modality)
  + Pulls values from **raptor\_list\_radiological** table.
  + Values for $type are Oral or IV to filter on **type\_nm** field value
  + Values for $modality are CT, MR, and NM
    - CT returns rows where ct\_yn=1
    - MR returns rows where mr\_yn=1
    - NM returns rows where nm\_yn=1
  + Returns simple array of values in **option\_tx** field.
* Function getSedationOptions($type, $modality)
  + Pulls values from **raptor\_list\_sedation** table.
  + Values for $type are Oral or IV to filter on **type\_nm** field value
  + Values for $modality are CT, MR, and NM
    - CT returns rows where ct\_yn=1
    - MR returns rows where mr\_yn=1
    - NM returns rows where nm\_yn=1
  + Returns simple array of values in **option\_tx** field.
* Function getContrastOptions($type, $modality)
  + Pulls values from **raptor\_list\_contrast** table.
  + Values for $type are Oral or IV to filter on **type\_nm** field value
  + Values for $modality are CT, MR, and NM
    - CT returns rows where ct\_yn=1
    - MR returns rows where mr\_yn=1
    - NM returns rows where nm\_yn=1
  + Returns simple array of values in **option\_tx** field.
* Function getAtRiskMedsKeywords()
  + Pulls values from **raptor\_atrisk\_meds** table.
  + Returns simple array of **keyword** field values.

## Class BoilerplateTextHelpers

This class provides values for content for boilerplate helper interfaces. It is implemented in the **data\_boilerplatehelpers.php** file.

### Functions to Read Data

* Function getTextHelpers($type)
  + Pulls values from the **raptor\_boilerplate\_helper** table.
  + Values for $type are PROTOCOL, EXAM, or MESSAGE
    - If PROTOCOL then only rows with 1 in protocol\_yn are included
    - If EXAM then only rows with 1 in exam\_yn are included
    - If MESSAGE then only rows with 1 in message\_yn are included
  + Returns array of arrays with following structure…
    - Key in first array is value of **category\_nm** field
    - Values in the array associated with the key are the **snippet\_tx** values for that key.

# Global Variables

The **raptor\_omega** theme gets relevant values via the global variables shown in Table 8Table 1. They are only populated as needed by the application context.

*Table 8 - Global Variables*

|  |  |
| --- | --- |
| Variable Name | Value Description |
| $raptor\_context | Contains instance of Context. To get the user information, call it like this…   * $raptor\_context->getUserInfo() |
| $raptor\_worklist\_rows | Contains result of getWorklistRows() |
| $raptor\_protocoldashboard | Contains result of getDashboardDetails() |
| $raptor\_protocol\_content | Contains arrays of arrays for the protocol page and its tabs. There are three top level named sections in the array as follows...   1. Reference 2. ContraIndications 3. Input   Named contents inside of section called "**Reference**" are as follows…   * ["**OrderOverview**"] has getOrderOverview() * ["**VitalsSummary**"] has getVitalsSummary() * ["**MedicationsDetail**"] has getMedicationsDetail() * ["**VitalsDetail**"] has getVitalsDetail() * ["**AllergiesDetail**"] has getAllergiesDetail() * ["**ProcedureLabsDetail**"] has getProcedureLabsDetail() * ["**DiagnosticLabsDetail**"] has getDiagnosticLabsDetail() * ["**DoseHxDetail**"] has getDoseHxDetail() * ["**PathologyReportsDetail**"] has getPathologyReportsDetail() * ["**ReportsDetail**"] has getSurgeryReportsDetail() * ["**ProblemsListDetail**"] has getProblemsListDetail() * ["**NotesDetail**"] has getNotesDetail() * ["**RadiologyReportsDetail**"] has getRadiologyReportsDetail() * ["**Graph**"]["**Thumbnail**"] has data for the thumbnail graph * ["**Graph**"]["**Labs**"] has data for the labs graph * ["**Graph**"]["**Vitals**"] has data for the vitals graph * [“**AtRiskMeds**”] has getAtRiskMedsKeywords() result.   Named contents inside of section called “**ContraIndications**” are as follows…   * TBD |
|  |  |

# Local RAPTOR Database

All interaction with the local raptor MySQL database should be abstracted through functions calls of the raptor data layer module. The tables of the MySQL database are as follows:

1. raptor\_atrisk\_meds
2. raptor\_boilerplate\_helper
3. raptor\_contraindication\_rule
4. raptor\_cprs\_codes
5. raptor\_group
6. raptor\_group\_anatomy\_keyword
7. raptor\_group\_modality
8. raptor\_group\_service
9. raptor\_list\_contrast
10. raptor\_list\_hydration
11. raptor\_list\_modality
12. raptor\_list\_radioisotope
13. raptor\_list\_sedation
14. raptor\_list\_service
15. raptor\_protocol\_keywords
16. raptor\_protocol\_lib
17. raptor\_protocol\_template
18. raptor\_role
19. raptor\_schedule\_location
20. raptor\_schedule\_track
21. raptor\_ticket\_exam\_notes
22. raptor\_ticket\_exam\_settings
23. raptor\_ticket\_collaboration
24. raptor\_ticket\_lock\_tracking
25. raptor\_ticket\_protocol\_notes
26. raptor\_ticket\_protocol\_settings
27. raptor\_ticket\_protocol\_settings\_replaced
28. raptor\_ticket\_qa\_notes
29. raptor\_ticket\_tracking
30. raptor\_ticket\_workflow\_history
31. raptor\_user\_anatomy
32. raptor\_user\_group\_membership
33. raptor\_user\_modality
34. raptor\_user\_profile
35. raptor\_user\_service

# Revision History

|  |  |  |
| --- | --- | --- |
| When | Who | Description |
| 20140111 | Frank Font | Initial draft based on content from POC and architecture of phase II RAPTOR application. |
| 20140217 | Frank Font | Added mention of “raptor” namespace and expectation that values are returned in associative arrays. Renamed boilerplate to dashboard for clarity. Also split requested date and scheduled date in the dashboard. Also added getStaticModeName function. Simplified Dashboard Content. Created section for global variables. |
| 20140222 | Frank Font | Additional global variable |
| 20140301 | Frank Font | Added getVitalsSummary |
| 20140315 | Frank Font | Removed getTicketType based on SME discussion last week that diagnostic/image guided criteria will be resolved via protocol and contra indication settings rather than derived from order information. |
| 20140330 | Frank Font | Added Snippet array member and renamed Date to DiagDate in one function so it matches implementation. |
| 20140406 | Frank Font | Updated to capture choice of tracking ID consisting of site ID and MDWS IEN value. Also updated row 1 of table 7 and changed Date to ReportDate in two functions. |
| 20140418 | Frank Font | Added helper functions and UserInfo class. Added schedule info to worklist data. |
| 20140419 | Frank Font | Added data\_user, data\_schedule, data\_images, and data\_protocollibrary sections. |
| 20140424 | Frank Font | Added more content related to local database and related functions. Also documented new graph entries in global **raptor\_protocol\_content** variable. |
| 20140510 | Frank Font | Added CPRS Code to Table 4 and Table 6, and added Imaging Type to Table 4. Added getUserName function to UserInfo class. Enhanced content of getProtocols array. Revised and enhanced API of the UserInfo class. Updated listing of RAPTOR database table names. |
| 20140513 | Frank Font | Added function to UserInfo class. |
| 20140516 | Frank Font | Defined raptor\_atrisk\_meds and getAtRiskMedsKeywords function. |
| 20140520 | Frank Font | Updated worklist results information. |
| 20140531 | Frank Font | Enhancement to getAllergiesDetail() structure |
| 20140605 | Frank Font | Defined two more fields for dashboard (mpiPid and mpiChecksum). |
| 20140606 | Frank Font | Added fields needed by VIX into result from getRadiologyReportsDetail(). Also defined QA ticket state. |
| 20140607 | Frank Font | Assignment is now an associative array. |
| 20140609 | Frank Font | Added Ranking value as 18th item of the wroklist row. |
| 20140610 | Frank Font | Added snippet and detail for another item and removed verified from result because I do not see a verification date or marker in the MDWS result. Added snippet/detail to ObservedHistorical field because it shows from MDWS comment. |
| 201406xx | Frank Font et all | Updates via Google Docs |