Argos + Luminis Portal

API Developer's Guide

by Matt Rapczynski
Database Analyst
Foothill-De Anza Community College District

https://github.com/mrapczynski/fhda-argos-api

The following document is a modified open source edition intended for public distribution with the FHDA Argos API project available on GitHub.

Preface

This guide has been designed to help you understand how to quickly and securely deploy an Argos report to the functional user community using portal.

Portal users who have Banner accounts are also provided a “Reports” tab in their portals. On this tab is a channel called “My Reports”, and it provides a personalized list of reports that users can run on their own without needing to meet specific hardware or software requirements. In addition, for most reports, IT staff will not need to be called upon to assist unless the user has a specific question or encounters a technical problem.

The goal is that functional users can get self service access to their data through reports designed for them while simultaneously reducing the number of requests for data that ultimately land on the desks of IT staff for processing.

This guide assumes that you have access to a Banner database, have a working knowledge of SQL, and that you are an Argos administrator. Additionally a graphical SQL client such as Oracle SQL Developer, Toad, PL/SQL Developer, or Aqua Data is *highly* recommended to make deploying a report easier. If nothing else, SQL*Plus will work too.

API Change Log

Date	Change/Event
May 26, 2011	Guide created. *API integration version 1 (V1).*
July 12, 2011	**Significant Changes** *API integration Version 2 (V2)* • New report categorization • New parameters for configuring fields • Added dynamic data binding *(page 11)* • Improved field validation options including new “templates” *(page 12)* • New report usage metrics *(page 13)* • New personalized reporting *(page 14)* • Deprecated specific components from V1 • Table ARGOS_API_ROLES (Step 3 has been entirely replaced in this document. Read over the changes carefully.) • Column FIELD_NOSEPERATOR • Column FIELD_USEBINDVAR • Miscellaneous layout changes throughout entire document
March 22, 2012	• **Open source** version created for Evisions Conference 2012

Introduction

The API functionality of Argos essentially provides a secure, invisible “pipeline” which enables executing a report and returning the finished result in a user-friendly format such as Adobe PDF or Excel. It is up to the Evisions client doing an Argos implementation to provide the user interface that takes advantage of this reporting pipeline.

Since the API is accessible exclusively through a web client using HTTP (or HTTPS), it assumed by Evisions that developers will create their own HTML based forms to submit user entered values as parameters to Argos. It is also up to the implementor of Argos to provide their own security model so that unauthorized requests for reports are not sent through the web for processing.

It is understood that developers have varying amounts of skill using HTML, and this presents a problem when the goal is to create a unified, consistent service for reporting. Additionally, creating an HTML document for each report to be executed, designing the parameter entry form for users, and securing that form can create a significant burden on IT staff both to create the form and maintain it in the years to follow. To satisfy all of the requirements mentioned above, Foothill-De Anza has created its own customized interface for developers to deploy reports to users for access within their portal accounts.

Benefits of Using Portal:

- Personalized content for the user community.
- Rich environment for generating interactive forms, and for providing helpful resources to users *(report library, simplified access to Argos client software).*
  - IT staff **not required** to become proficient HTML, CSS, or JavaScript programing skills to deploy an interactive form.
- No extra accounts required for a partner system - single sign-on aware.
- Leverage both Luminis and Banner security models to control report distribution without re-inventing the wheel.
- Cross-platform - reports accessible through any operating system (Windows, Mac, and Linux) and modern browser software (Firefox, Safari, Chrome, IE7-9+).
  - Argos client software **not required** for the greater user community. Only power users desiring to create reports will need to access the full development environment.

Getting Started

Deploying a report using the portal integration process involves primarily working in the Banner database. Rather than expect to design an HTML page *for each* report to be delivered through the web, a set of database tables has been created that allow you to describe the common attributes of a report, its security, and the fields (parameters) it contains.

Once this definition is complete, an HTML form with all of the correct formatting will be created for you (or really for the user) on demand when the report is accessed through the portal. We do not store pre-built HTML documents on any server, and instead we use server side technologies to read the records you insert into the API tables to generate a form dynamically.

Detailed table structures will be laid out in subsequent pages.

Table Name	Description
ARGOS_API_FORMS	Every report to be delivered on the portal starts here. One record per report. Common attributes such as a description, the API key, and file formats that this report generates are set up in this table.
ARGOS_API_FIELDS	The individual fields that will be displayed to the user to input parameters are defined in this table. One record per field.
ARGOS_API_OVERRIDES	In some instances, a report that has been certified for web delivery cannot be associated with a large enough population that justified the use of a role or security class. You can grant web access to any specific employee by mapping their CWID to a report in this table.
ARGOS_API_RPTSEC	Define more than one security class or Luminis role that will have access to execute this report.

Deploying a report typically will follow this sequence of steps after the Argos report itself is completed:

1. Insert a record into table ARGOS_API_FORMS. Give the report a unique identifier, and describe the report. At this step is where you define how the users' portal session will talk to Argos using the "API keys", and the security which determines whether this report will be accessible to a user.

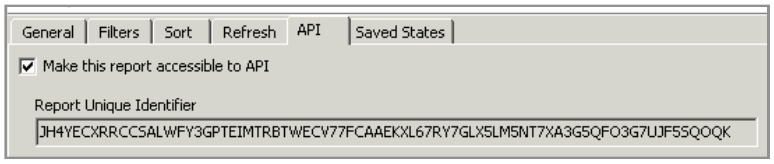
2. Insert record(s) into table ARGOS_API_FIELDS for each field. The users' input is passed directly to the Argos server for report execution. Your fields should be similar, if not identical, to the structure/design of the data block.

3. Optionally, configure report security to include more than one security class (table ARGOS_API_RPTSEC) or specific user grants (table ARGOS_API_OVERRIDES).

4. Done! It's really simple. After inserting records into the database, your report is immediately available online and ready for "go-live" testing.

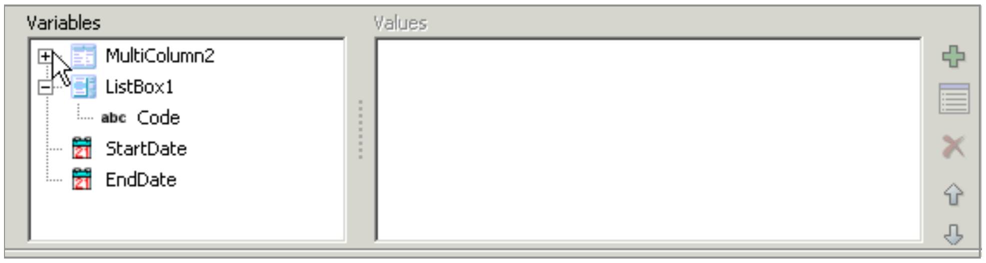
Step 1 - Define the Report

As you see in the summary of directions on the previous page, the first step is to define the report to be deployed in the ARGOS_API_FORMS table. Use an SQL insert statement, and map values to the appropriate column by following the table spec outlined below:

ARGOS_API_FORMS	
Column Name	**Description**
REPORT_NAME **Required** VARCHAR2 - 24 Char	A unique identifier for this form. This column is part of the foreign key in the fields table. It is case-sensitive, and ALL CAPS is recommended. Do not use punctuation characters other than the underscore.
REPORT_TITLE **Required** VARCHAR2 - 64 Char	“Pretty” title for this report. This will be visible to the users on the web. HTML tags are supported for enhanced formatting.
REPORT_APIKEY **Required** VARCHAR2 - 512 Char	Report identifier string generated by Argos when API access is enabled for a specific report. Copy and paste from the API tab of the report properties within Argos.  General Filters Sort Refresh API Saved States ☑ Make this report accessible to API Report Unique Identifier JH4YECXRRCCSALWFY3GPTEIMTRBTWECV77FCAAEKXL67RY7GLX5LM5NT7XA3G5QFO3G7UJF5SQOQK Don’t forget to setup the user and password. Since we apply security to Argos, this is **required.** Credentials will not be printed in this document.
REPORT_CATEGORY **Optional** VARCHAR2 - 96 Char	The “My Reports” channel for end users applies a GROUP BY query behind the scenes to create logical groups of reports. This value should be considered **case-sensitive.** Reports with a NULL category will be organized into a default group labeled “Specialty”.
REPORT_DESC **Optional** VARCHAR2 - 2048 Char	“Pretty” description for this report. This will be visible to the users on the web. HTML tags are supported for enhanced formatting.
REPORT_ROLE **Optional** VARCHAR2 - 96 Char	The name of a Luminis role, or a Banner security class, which defines the population of users who may execute this report. The security logic will determine automatically what type of group you provided. If you leave this field NULL, then only those granted an override can see this report. Also see page 9.
REPORT_PDF_FLAG **REPORT_XLS_FLAG** **REPORT_CSV_FLAG** **Optional** VARCHAR2 - 1 Char	Configure which file formats this report targets. All three fields take a ‘Y’ or ‘N’ value (case-sensitive). Based on the values you input, the user will only see download options for the formats marked ‘Y’. *Certain reports such as banded do not support CSV. Refer to Argos documentation for greater detail.*
REPORT_ENABLE_PIDM **REPORT_ENABLE_CWID** **Optional** VARCHAR2 - 1 Char	Either field accepts a ‘Y’ for enabled, or ‘N’ for disabled (case-sensitive). ENABLE_PIDM will pass the PIDM value back to Argos, and ENABLE_CWID passes the user CWID back to Argos. This feature is explained in greater detail on page 14.

Step 2 - Define the Fields

Next, to define the fields of the form, insert records into the ARGOS_API_FIELDS table. Each record represents a different field for a user enterable parameter of the Argos data block associated with the report.

ARGOS_API_FIELDS	
Column Name	**Description**
REPORT_NAME **Required** VARCHAR2 - 24 Char	The unique identifier of the form that this field belongs to. Must exactly match the entry in ARGOS_API_FORMS to establish a relationship between fields and the form they should be rendered with.
FIELD_SEQNO **Required** NUMBER	Enter a numeric value to identify this field - sequential numbers work best. The order that the fields are displayed to the user will be based on the ascending sequence of all the defined fields. This value is also used a unique identifier for each field.
FIELD_TYPE **Required** VARCHAR2 - 12 Char	One of three possible values. Determines what kind of HTML widget to render. 1. **INPUT** - Standard single-line text box 2. **VALLIST** - List box widget containing a fixed set of values 3. **BOOLEAN** - A simplified list widget that permits sending a 'Y' or 'N' value to Argos ('Y' = Yes/True, 'N' = No/False). Cannot be modified.
FIELD_LABEL **Required** VARCHAR2 - 48 Char	"Pretty" field label that will be seen by users on the web. HTML tags are supported for enhanced formatting.
FIELD_PARAM_NAME **Required** VARCHAR2 - 24 Char	**Important.** The name of the variable in the Argos data block should be put here. The list of possible variable names is available in the API tab on the Edit Report dialog through Argos. It is very important to take note of both a variable name, and in some instances variables are nested as children of a parent UI widget in the data block.  Variables Values MultiColumn2 ListBox1 Code StartDate EndDate In the example above, the correct variable names are "StartDate", "EndDate", "ListBox1.Code", etc…
FIELD_COMMENTS **Optional** VARCHAR2 - 256 Char	Extra notes or a detailed description of this field. This will be visible to the users on the web. HTML tags are supported for enhanced formatting.

ARGOS_API_FIELDS	
Column Name	**Description**

FIELD_SQL
Optional
VARCHAR2 - 1024 Char
	Applies only to fields whose type is set to "VALLIST".
	
	To simplify populating list boxes with values direct from the Banner database, you can design an SQL statement that will be executed when the form is rendered for the user. The SQL statement you provide must follow a few simple rules:
	
	1. You must select only TWO columns. The first column should represent the REAL database value that should be sent back to Argos. The second column should be any kind of value that is a "pretty" identifier for what that value represents. Only the description will be visible to the user.
	2. The SQL statement is executed using Oracle dynamic SQL, and so any database-level rules set by Oracle must also be adhered to.
	http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14251/adfns_dynamic_sql.htm
	
	Example:
	`SELECT stvterm_code, stvterm_desc`
	`FROM stvterm`
	`ORDER BY stvterm_code DESC`

FIELD_BINDING **Optional** VARCHAR2 - 96 Char	**Applies only to fields whose type is set to "VALLIST".** Dynamic data binding allows you to configure fields that auto-refresh based on data the user selects in another field. Use the FIELD_PARAM_NAME value from another field in this column to establish a relationship between two fields. When the value of the bound field changes, then the value list for this field will refresh from the database. There are important pre-requisites that need to be met for this function to work smoothly. Dynamic data binding is discussed in greater detail on page 11.
FIELD_REGEX **Optional** VARCHAR2 - 128 Char	**Applies only to fields whose type is set to "INPUT".** This field is provided to allow you to specify a regular expression pattern that will be tested against the value typed in by the user *prior to submitting* a report request to Argos. Data validation is discussed in greater detail on page 12.
FIELD_REGEX_ERRMSG **Optional** VARCHAR2 - 1024 Char	**Applies only to fields whose type is set to "INPUT".** If a field fails validation, then the message stored in this column will be presented to the end user. Supply useful information or directives that help the user correct their mistake.
FIELD_REQUIRED **Optional** VARCHAR2 - 1 Char	Set this field to a 'Y' (yes, required) or 'N' (no, not required) to enforce that the user supply a value before the report is sent for execution.
FIELD_DISABLED **Optional** VARCHAR2 - 1 Char	Set this field to a 'Y' (field hidden) or 'N' (field visible) to hide a field on the form. It will not be rendered if this field is set to 'Y'. Helpful for testing specific fields.

ARGOS_API_FIELDS	
Column Name	**Description**
FIELD_MULTIPLESEL **Optional** VARCHAR2 - 1 Char	**Applies only to fields whose type is set to “VALLIST”.** Set this field to ‘Y’ (yes, multiple selections permitted) or ‘N’ (multiple selections not permitted). This setting will also change the appearance of the widget. Fields with multiple selections will display as a list box. Otherwise the default appearance is a single-select combo box.
FIELD_INITMSG **Optional** VARCHAR2 - 1 Char	**Applies only to fields whose type is set to “VALLIST”.** When used with dynamic data binding, and the list of values is empty, then the value supplied in this column will be used as an initial prompt visible to be user. Use this to provide short, helpful directions to the end user.

Step 3 - Configuring Report Security

Deploying to a Single Security Class

Most reports will be deployed to a specific group of people easily defined by a single security class. If this is case, then all of the work is done for you once that class has been specified in the REPORT_ROLE column from step 1.

Deploying to Multiple Security Classes

Not every report will be adequately addressed with respect to its security by only one security class. If this scenario applies to your report, then adding records to the ARGOS_API_RPTSEC table will solve the problem. Each record inserted binds a security class to a report, and multiple records can exist per report. These relationships will be applied when a user visits their “My Reports” channel.

ARGOS_API_RPTSEC	
Column Name	**Description**
REPORT_NAME **Required** VARCHAR2 - 24 Char	The report identifier (matches REPORT_NAME from ARGOS_API_FORMS).
CLASS_NAME **Required** VARCHAR2 - 30 Char	The name of the security class whose members will be permitted to execute this report through their portal accounts.

User Overrides

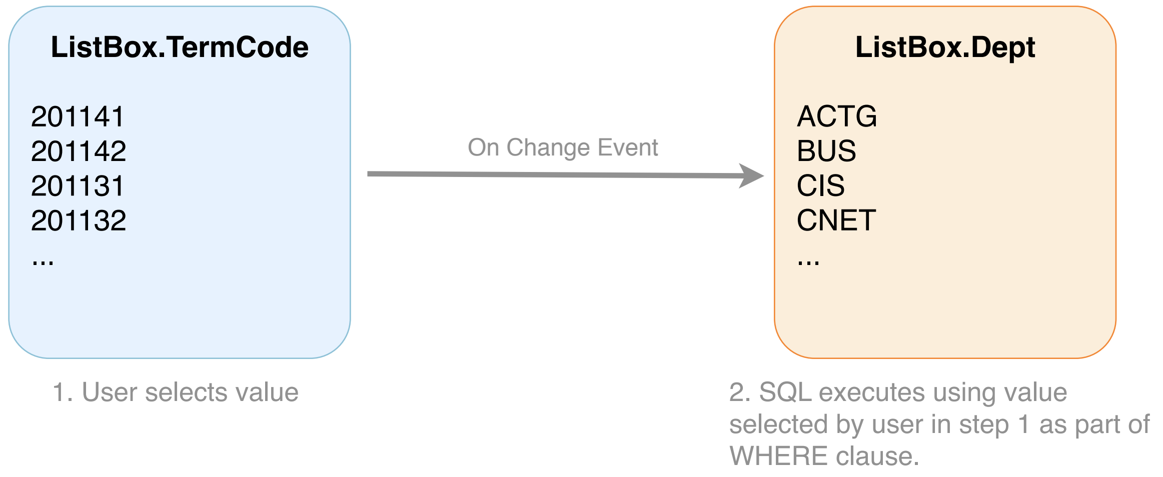
You are likely to find when working with specific staff members that their job role is not accurately represented by any of the available Banner security classes. In these instances, you can create a relationship between a person and a specific report. In doing so, the report becomes immediately available on their portal without being going through the process of being checked for a group membership.

ARGOS_API_OVERRIDES	
Column Name	**Description**
OVERRIDE_ID **Required** VARCHAR2 - 12 Char	The CWID of the employee who will be granted a special exception.
OVERRIDE_REPORT **Required** VARCHAR2 - 24 Char	The report identifier (which matches REPORT_NAME from ARGOS_API_FORMS) that they are permitted to access through the portal.

Developer Tip: This is also a highly effective method to permit private testing of a report without releasing it on accident to the larger user community. You can grant yourself an override as a developer and then test your work using your own personal portal account, or include a few staff members who are helping direct the testing of the report. When you’re ready to release your certified work to a wider audience, update the REPORT_ROLE column in ARGOS_API_FORMS or add more override records as needed.

Enabling Dynamic Data Binding

Dynamic data binding allows you to replicate one of the more elegant features from the Argos thick client interface right inside the web browser. That feature is the behavior to update a list widget when the user changes the selection in another "bound" list widget. No JavaScript/AJAX work required - the underlying mechanics are done for you.



The ability of the API to replicate this behavior for the user applies only to the use of value lists. Presently, input fields cannot be bound.

There are two important requirements to meet if you want to enable this behavior in your forms:

1. **Develop an SQL statement that includes a single bind variable for the FIELD_SQL column.**
    1.1. You can name the bind variable anything you want. What matters the most is that it exists. A field configured with a binding will expect to execute with a bind variable. If that bind variable does not exist, then an Oracle error will occur.
    1.2. The Oracle rules and security features for dynamic SQL will be enforced.
2. **Bind the destination field to a source field that will generate the change events.**
    2.1. To do this, copy the FIELD_PARAM_NAME of the source field into the FIELD_BINDNG column for the destination.
    2.2. In the example above ListBox.TermCode is the source, and ListBox.Dept is the destination.
    2.3. Thus, ListBox.Dept is *bound* to ListBox.TermCode. ListBox.Dept will receive the value selected in ListBox.TermCode, and pass that value into the bind variable.

FIELD_SQL Example:

Source Field:
```
select stvterm_code, stvterm_code || ' - ' || stvterm_desc from stvterm
```

Destination Field:
```
select distinct ssbsect_subj_code from ssbsect where ssbsect_term_code = :TermCode
```

Using Data Validation

Bad or incorrect data can drastically alter report accuracy, or generate confusing Oracle errors. Data validation works to reduce the chances of user generated errors by catching problems before the report is run. As the developer, you have control over the rules used to determine whether data supplied by the end user is good or bad for each field. Those rules are defined using regular expressions.

Validations are performed client side in the web browser. This requires that you use regular expression patterns that conform to the ECMAScript standard for JavaScript. Note that actual JavaScript knowledge is *not required*, but we must conform to this standard since it is widely available in all major web browsers.

If you are familiar with regular expressions, then review this reference material maintained by Mozilla: https://developer.mozilla.org/en/JavaScript/Guide/Regular_Expressions (Writing a Regular Expression Pattern)

There are valuable web apps available to help you quickly test and identify problems with regular expressions that you have designed. A recommended tool can be accessed here:
http://regexpal.com

Several built-in templates are available to you. Using a template is simple: enclose the template name in << and >> brackets, and insert that value into the FIELD_REGEX column. Optionally, provide an error message in the FIELD_REGEX_ERRMSG column for the user if validation fails.

Built-in Regular Expression Templates

Template Name	Description
ALPHA_ONLY	Enforce that the input contains only A-Z or a-z characters Pattern: `^[A-Za-z]{1,}$`
ALPHA_UPPER_ONLY	Enforce that the input contains only A-Z characters Pattern: `^[A-Z]{1,}$`
ALPHA_LOWER_ONLY	Enforce that the input contains only a-z characters Pattern: `^[a-z]{1,}$`
NUMERIC_ONLY	Enforce that the input contains only 0-9 numbers, and a period separator Pattern: `^[0-9]{1,}$`
ALPHANUM_ONLY	Enforce that the input contains only A-Z, a-z, and 0-9 characters Pattern: `^[A-Za-z0-9]{1,}$`
ALPHA_SEARCH	Enforce that the input contains only A-Z, a-z, and must be flanked by a wildcard character for the LIKE operator (similar to Banner search behavior) Pattern: `(^[%]{1}[A-Za-z]{1,}$)\|(^[A-Za-z]{1,}[%]{1}$)`
DATE_SHORT	Enforce that the input matches a short hand date in the format of MM/DD/YYYY. *Example:* 1/1/2011 will match, but Jan 1, 2011 and 1/1/11 will not. Pattern: `^[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}$`

Usage Metrics

Anonymous statistics of report usage through the portal API are now collected as of version 2. This information is stored in the table ARGOS_API_METRICS.

The following is collected each time a user executes a report:

1. The report identifier (REPORT_NAME)
2. The Oracle date stamp when the report was executed, including the minute and second values
3. The entire query string sent to the Argos MAPS server (the user entered data block parameters and configuration values including report format and API key)
4. The entire user agent string sent by the user browser which includes the platform, browser vendor, and type of device (if mobile) that was used to execute the report.

This information will be available both as a statistical materialized view (accessible via SQL query), and an Argos report that can be printed for analysis.

Personalized Reporting

Users love viewing data that is tailored to their specific needs or job description. Using creative SQL applied at the data block level, you can deliver personalized reports to the user who runs them with the API without hardcoding their identity, or creating complicated schedule routines.

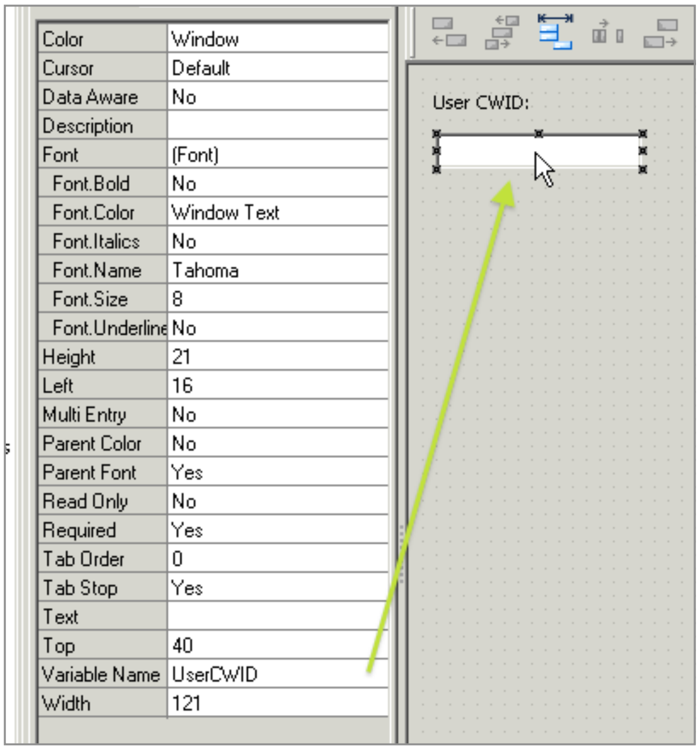
In page 5 of this guide, reference was made to two columns of the ARGOS_API_FORMS table that are available. They are REPORT_ENABLE_PIDM and REPORT_ENABLE_CWID. By setting one of these columns to enabled using an uppercase ‘Y’ character, the auto generated HTML form will include the identity of the person executing the report. This identity value will then be sent to Argos as a data block parameter along with the rest of the fields you defined.

You can include this in your data block by just adding an extra edit box widget to the form designer within Argos. It is very important that you adhere to the standard for naming this widget.

- If you are enabling passing of the CWID identity, then set the “Variable Name” of the edit box to “UserCWID”

- If you are enabling passing of the user PIDM as the identity, then set the “Variable Name” of the edit box to “UserPIDM”

Once the edit box has been created, then you can reference it in the SQL as part of the data block. The API will automatically populate this variable when the report is run from the web.



Color	Window
Cursor	Default
Data Aware	No
Description	
Font	(Font)
Font.Bold	No
Font.Color	Window Text
Font.Italics	No
Font.Name	Tahoma
Font.Size	8
Font.Underline	No
Height	21
Left	16
Multi Entry	No
Parent Color	No
Parent Font	Yes
Read Only	No
Required	Yes
Tab Order	0
Tab Stop	Yes
Text	
Top	40
Variable Name	UserCWID
Width	121