

Application Programming Interface

This documentation covers the functionality of the ExtraView Programming Interface (CLI). This is a REpresentational State Transfer (RESTful) API

This document is intended for the experienced computer user who has a good understanding of either the UNIX, Linux, or Microsoft Windows environments from which they will use the Application Programming Interface (API). To take maximum advantage of the features offered by the API, knowledge of web-based technologies are helpful.

If you intend to modify the behavior of ExtraView with user custom programming, you will need to be skilled in coding with the Java language. In addition, ExtraView administration skills are required to configure many of the functions offered.

Downloadable PDF

[The Application Programming Interface Guide is downloadable as a single PDF by clicking here.](#) You will need the [Adobe Acrobat Reader](#) to view this.

Related API Guides

- CLI - Command Line Interface
- Web Services Interface

The key features of the RESTful API are:

- Insert, update and delete records in the ExtraView database from remote applications
- Search the ExtraView database and return a set of records defined in a query
- Export information from the ExtraView database for input to a data warehouse
- Upload and download file attachments to and from the ExtraView database
- Provide limited administration access to create metadata and to manage user accounts
- Show the names of fields within the ExtraView database to which the user has access

Concepts

The REStful API is a set of programmable HTTP calls. These calls operate on the ExtraView application to return data and metadata to the calling program or to update items within the ExtraView database.

API command summary

ADD_ATTACHMENT	uploads a file attachment to the ExtraView database
ADD_FIELD_LIST	provides an ordered list of fields that are used to insert records
ADD_USER_TO_GROUP	adds an existing user to a user group
ALLOWED_LIST	gives the list of allowed values for a key
CUSTOM	invokes a call within the CLI user exit in the UserCustom java class
DEBUG	alters the debug level of messages sent to the application server log
DELETE	deletes existing issue(s) from the database

DELETE_USER	deletes an existing user
EDIT_FIELD_LIST	provides an ordered list of fields that are used for update
FIELDS	displays a list of available fields and their screen names
FILL_IN	uses a template and fills in the values with the parameters provided
GET	retrieves a single record from the database
GET_APP_DEFAULT	retrieve an ExtraView behavior setting (deprecated)
GET_AREAS	retrieves a list of the areas in use from the database
GET_ATTACHMENT	downloads a copy of a file from an issue attachment
GET_BEHAVIOR_SETTING	retrieve an ExtraView behavior setting (supercedes GET_APP_DEFAULT)
GET_FIELDS	download a specific field or fields from an existing issue
GET_HEARTBEAT	return a status indicator to show that ExtraView is alive
GET_LOG	returns the contents of the application server log
GET_PROJECTS	retrieves a list of projects for a given area from the database
GET_REPORTS	retrieves a list of the reports currently available to the user
GET_ROLES	retrieves the available roles for a specified user from the database
GET_TITLE	retrieve data dictionary information on a field
GET_USERS	retrieve a list of user names from the database
GET_VALID_META_DATA	retrieve the metadata for an issue
HISTORY	retrieve all records from a specified point in time
IMPORT_ALLOWED_VALUES	import allowed values from a file into ExtraView
INSERT	inserts new records into the ExtraView database
INSERT_USER	create a new user account
INSERT_XML	inserts a new record into ExtraView from XML formatted input data
ITEM_EXISTS	checks whether an issue exists in the database
LIST_ATTACHMENT	retrieves a list of file attachments against a single issue
RUN_REPORT	runs a report that has been defined within ExtraView
SEARCH	provides a general search and retrieval mechanism
SEARCH_FIELD_LIST	provides a list of fields that can be searched for
SET_AREA_PROJ	sets the working area and project
SET_ROLE	sets the working role of the user
UPDATE	updates existing records in the ExtraView database
UPDATE_USER_PASSWORD	updates a user's password
USER_FIELD_LIST	displays an ordered list of fields for the user record

USER_GROUP_LIST	displays a list of user roles ordered by the user role title
VERSION	returns the version number and build number of ExtraView

Authentication

All API commands require an active ExtraView username and password unless anonymous access is enabled with the behavior setting named `ALLOW_ANONYMOUS_API_ACCESS`. It is sometimes useful to allow anonymous access to ExtraView. Most commonly, this is used to provide a web page that you have designed to perform transactions, or to query the ExtraView database, not to need a hard-coded username and password.

To preserve a secure environment, it is necessary to perform the following steps to set up anonymous access:

1. Within the ExtraView administration section, under the Systems Control tab and within the API Settings section, set the behavior setting named `ALLOW_ANONYMOUS_API_ACCESS` to have a value of YES
2. In the same section, set the application default named `ANONYMOUS_API_USER_ID` to the user ID that you will use for the anonymous access
3. All entries made in an anonymous fashion will use this user ID to log their activities
4. Ensure that you only give security privileges to the user group to which this user ID belongs that is in keeping with the fact that this user is used to log all activities. Normally this means that this user group will have limited access to fields, especially in write and query modes
5. It is recommended that when you allow users to enter issues in an anonymous fashion, that you have ExtraView fields that capture their names and other contact details

Once you have set up anonymous access, the parameters named `user_id` and `password` in all API commands become optional. If they are provided, they must be valid combinations of user names and passwords.

Requests Passed to the API

To create an API call, the user creates an HTTP request (HTTPRequest object) and submits this to the server. This request can be composed within virtually any computer language. Each HTTP request is composed of the following parts:

- The server domain name, including the path to the servlet where ExtraView is running. For example, `http://www.myserver.com/evj/ExtraView`
- The class and method which accesses the API. Usually this is `ev_api.action`
- The `user_id` and password for the user, passed as name / value pairs
- The `statevar`, passed as a name / value pair. This parameter defines the specific API call being made
- A list of name / value pairs providing the parameters and their values for the API call, and which are necessary or optional to execute the API call. Note that the user is responsible to provide a correct list of parameters for each call. If the API does not recognize a parameter, then it is ignored and no warning is given to the user.

If you are creating your own HTTPRequest object from a programming environment to call the ExtraView API, then you must also set the standard parts of the HTTPRequest object, such as the `USER_AGENT`. There will most probably be standard libraries for your environment to assist in doing this.

XML Data Returned From API Calls

Much of the data returned by an API call is in XML format. This has some significance to the user of the

API, in that Extraview's XML data may embed your own XML within its results. To accommodate this, ExtraView uses Base64 encoding whenever it sees XML data returned from the application to ensure that the XML returned by API commands through the API must be well-formed. This means that the contents of a CDATA string must not contain the character string "]]>", because that is the end sentinel for a CDATA section. So, if the original data contains this string, there must be some way to escape the data. For easy recognition of an escaped CDATA string, ExtraView prepend the characters %25S to the front of the string. These characters are merely a sentinel and are not part of the output string. The encoding used for the rest of the CDATA string is called Base64, and algorithms for encoding/decoding are widely available. Furthermore, ExtraView ensures that the %25S sentinel string does not appear in the CDATA raw character string by encoding any CDATA raw character string to Base64 as well. It is the responsibility of the receiver, therefore, to test each CDATA section for the sentinel characters %25S at the beginning of the CDATA, and, if present, perform the Base64 decode function on the remainder of the character data to get the raw character values in the field.

Server-side Templates

Server-side templates can be defined for many API commands. These templates allow you to control the presentation of the output from the API command. Most commonly this is used to allow the administrator to integrate ExtraView with their company's own web site. For example, a page within your company's web site can perform a search of the ExtraView database and return the results formatted with the same look and feel as the web site.

Templates allow you to use "tags" that are substituted at runtime, with the actual value in a record. For example, the tag `__STATUS__` refers to the value of the STATUS field of the current record. Tags are available for most fields in the data dictionary. In addition, if you place a data dictionary field on a template, it must exist on the Detailed Report layout inherited by the specific Business Area and Project. The user must also have read permission to the field.

[See this page](#) for full details of server-side templates.

Repeating Row Records and the API

Output from ExtraView using repeating rows can be turned off, to simplify the XML returned from a command. To use repeating rows within the API, you must set the behavior setting named `MULTI_RELEASE_XML` to a value of YES within the administration section.

The following API commands support repeating rows:

- GET
- GET_FIELDS
- HISTORY
- INSERT – note that you can only insert a single repeating row when inserting an issue, but you can then update that issue with additional repeating rows
- RUN_REPORT
- UPDATE – note that you can only update a single repeating row with a single update command.

The commands that return XML will do so as shown in the following example:

```
<REPEATING_ROWS TITLE='Repeating Rows' REPEATING_ROWS='1'>
<REPEATING_ROW REPEATING_ROW_ID='1073779532'>
<FRED_RR TITLE='Repeating row fred'>
<![CDATA[a new fredrr]]>
</FRED_RR>
<RELEASE_CHILD_STATUS TITLE='Branch Status'>
<![CDATA[Assigned]]>
</RELEASE_CHILD_STATUS>
```

```

<RELEASE_REQUESTED_RELEASE TITLE='Requested Release'>
<![CDATA[4.0.3.14]]>
</RELEASE_REQUESTED_RELEASE>
<RELEASE_COMMITTED_RELEASE TITLE='Committed Release'>
<![CDATA[4.0.3.14]]>
</RELEASE_COMMITTED_RELEASE>
</REPEATING_ROW>
<REPEATING_ROW REPEATING_ROW_ID='1073779542'>
<FRED_RR TITLE='Repeating row fred'>
<![CDATA[a new fredrr2]]>
</FRED_RR>
<RELEASE_CHILD_STATUS TITLE='Branch Status'>
<![CDATA[Assigned]]>
</RELEASE_CHILD_STATUS>
<RELEASE_REQUESTED_RELEASE TITLE='Requested Release'>
<![CDATA[4.0.3.13]]>
</RELEASE_REQUESTED_RELEASE>
<RELEASE_COMMITTED_RELEASE TITLE='Committed Release'>
<![CDATA[4.0.3.13]]>
</RELEASE_COMMITTED_RELEASE>
</REPEATING_ROW>
<REPEATING_ROW REPEATING_ROW_ID='1073779552'>
<FRED_RR TITLE='Repeating row fred'>
<![CDATA[a new fredrr3]]>
</FRED_RR>
<RELEASE_CHILD_STATUS TITLE='Branch Status'>
<![CDATA[Assigned]]>
</RELEASE_CHILD_STATUS>
<RELEASE_REQUESTED_RELEASE TITLE='Requested Release'>
<![CDATA[4.2.2.7]]>
</RELEASE_REQUESTED_RELEASE>
<RELEASE_COMMITTED_RELEASE TITLE='Committed Release'>
<![CDATA[4.2.2.8]]>
</RELEASE_COMMITTED_RELEASE>
</REPEATING_ROW>
</REPEATING_ROWS>

```

The data structures here are:

1. Repeating Rows Element -- at most one per issue extracted:

```

<REPEATING_ROWS TITLE='Repeating Rows' REPEATING_ROWS='1'>
... (repeating row elements)
</REPEATING_ROWS>

```

2. Repeating Row Elements – as many as there are repeating rows in the issue extracted:

```

<REPEATING_ROW REPEATING_ROW_ID='nnnnnnn'>
... (repeating row field elements)
</REPEATING_ROW>

```

3. Repeating Row Field Elements – one for each field inside the repeating row:

```

<fieldTag TITLE='fieldTitle'>

```

```
<![CDATA[fieldData]]>
</fieldTag>
```

where –

fieldTag = data dictionary field name

fieldTitle = title of the field

fieldData = data contained in the field (in CDATA-encoded format)

Updating and Inserting Fields in Repeating Row Data

Previous to release 4.2.2.8, the API did not handle repeating row records, therefore the previous format for for update of fields within an issue is maintained for backward compatibility. An extension is made to allow for the insertion or the update of one or more specific repeating row fields as follows.

Updating Repeating Rows

To update an existing repeating row field, there must be a parameter named `PROBLEM_RELEASE_ID` in the update parameter set. This parameter may have multiple values, each one specifying a specific repeating row ID (i.e. the `ITEM_ID` of the repeating row within the issue). The field values associated with these repeating rows are specified in a parallel set of parameters, named for the field name within the repeating row. Each such parameter may be given multiple values, each of which maps to a repeating row as specified by the multiple values of `PROBLEM_RELEASE_ID`.

As an example, assume that there are 5 repeating rows in issue #100, numbered 201, 202, 203, 204, and 205.

Then with `PROBLEM_RELEASE_ID` set to the values (202, 204), the value of `FRED_RR`, a repeating row-resident field, might take on the values ("fred202", "fred204") in the input parameters.

Then, the repeating rows with the ID's of 202 and 204 are updated with the value of `FRED_RR` being set to "fred202" and "fred204" respectively. No other repeating rows would be affected by the update.

Inserting Repeating Rows

Inserting repeating rows is similar to updating repeating rows, except no `PROBLEM_RELEASE_ID` is given as a parameter. When no `PROBLEM_RELEASE_ID` is present, ExtraView will insert the remaining fields as a new repeating ROW.

You can insert multiple repeating rows with a single API call. For example, the following parameter string will insert three repeating row records, each within a single field named `FRED_RR`:

```
&FRED_RR=val1&FRED_RR=val2&FRED_RR=val3
```

Repeating Row Example

First, make sure you have set the behavior setting named `MULTI_RELEASE_XML` to YES. The following examples assume you have an ExtraView server located at <http://extraview.myserver.com/evj/ExtraView>.

Retrieving an Issue

We will first retrieve issue number 26788 from ExtraView. This issue has existing repeating rows.

```
http://extraview.myserver.com/evj/ExtraView/ev_api.action?
```

user_id=myname&password=mypass&statevar=get&id=26788

ExtraView returns XML data that includes a section about Repeating Rows. The XML is shown indented here for clarity.

```

... ..
<REPEATING_ROWS TITLE="Repeating Rows" REPEATING_ROWS="1">
<REPEATING_ROW REPEATING_ROW_ID="1073788412">
<RELEASE_SOURCE_CODE_BRANCH TITLE="Code Branch">
<![CDATA[ 4.3 ]]>
</RELEASE_SOURCE_CODE_BRANCH>
<RELEASE_CHILD_STATUS TITLE="Branch Status">
<![CDATA[ Assigned ]]>
</RELEASE_CHILD_STATUS>
<RELEASE_REQUESTED_RELEASE TITLE="Requested Release">
<![CDATA[ 4.3.1 ]]>
</RELEASE_REQUESTED_RELEASE>
<RELEASE_COMMITTED_RELEASE TITLE="Committed Release">
<![CDATA[ 4.3.1 ]]>
</RELEASE_COMMITTED_RELEASE>
</REPEATING_ROW>
<REPEATING_ROW REPEATING_ROW_ID="1073791532">
<RELEASE_SOURCE_CODE_BRANCH TITLE="Code Branch">
<![CDATA[ 4.4 ]]>
</RELEASE_SOURCE_CODE_BRANCH>
<RELEASE_CHILD_STATUS TITLE="Branch Status">
<![CDATA[ Open ]]>
</RELEASE_CHILD_STATUS>
<RELEASE_REQUESTED_RELEASE TITLE="Requested Release">
<![CDATA[ 4.4 ]]>
</RELEASE_REQUESTED_RELEASE>
<RELEASE_COMMITTED_RELEASE TITLE="Committed Release">
<![CDATA[ 4.4 ]]>
</RELEASE_COMMITTED_RELEASE>
</REPEATING_ROW>
</REPEATING_ROWS>
... ..

```

The important data you need is located at . This provides the unique identifier for the particular repeating row of the issue.

Updating an Issue

To update the field named RELEASE_CHILD_STATUS field within the repeating record field just retrieved from the example above:

```

http://extraview.myserver.com/evj/ExtraView/ev_api.action?user_id=myname
&password=mypass&statevar=update&id=26788&problem_release_id=1073788412
&RELEASE_CHILD_STATUS=6936

```

Inserting an Issue

To insert a new issue, including one repeating row:

```
http://extraview.myserver.com/evj/ExtraView/ev_api.action?user_id=myname
&password=mypass&statevar=insert&area_id=4&project_id=8&category=SOFTWARE
&short_descr=Testing&product_name=EVJAVA&module_name=DATABASE&RELEASE_CHILD_STATUS=6936
&description=blah
```

The presence of the field named RELEASE_CHILD_STATUS, as the only field in the example that has exists within a repeating row record, causes a new repeating row record to be created.

Server-Side Templates

Server-side templates can be defined for many API and CLI commands. These templates allow you to control the presentation of the output from the API command. Most commonly this is used to allow the administrator to integrate ExtraView with their company's own web site. For example, a page within your company's web site can perform a search of the ExtraView database and return the results formatted with the same look and feel as the web site.

Templates allow you to use "tags" that are substituted at runtime, with the actual value in a record. For example, the tag `__STATUS__` refers to the value of the STATUS field of the current issue. Tags are available for most fields in the data dictionary. In addition, if you place a data dictionary field on a template, it must exist on the **Detailed Report** layout inherited by the specific Business Area and Project. The user must also have read permission to the field.

TEXTAREA, LOGAREA and PRINTTEXT fields have special handling within ExtraView. This is because they can be broken down into three components, the text itself, the user's name who entered the text, and a timestamp. Each of these components can be accessed individually, as shown in the following example. Note that the field name itself must be included as a tag, although it does not display anything in the output. Therefore, an HTML fragment that might display the DESCRIPTION field may look like:

```
<TD>
__DESCRIPTION__
__DESCRIPTION.USER__ : __DESCRIPTION.TIMESTAMP__
<BR><BR>
__DESCRIPTION.TEXT__
<BR>
</TD>
```

The full explanation for each part of the field is as follows, where DDNAME is the data dictionary name of the field of display type TEXTAREA, LOGAREA or PRINTTEXT.

Field name	Explanation
<code>__DDNAME__</code>	A tag with the data dictionary name must be included in the template. No output occurs with this tag. It is a placeholder that ensures the remaining fields will be processed correctly
<code>__DDNAME.TEXT__</code>	This is used to return the body of text within the comments
<code>__DDNAME.USER__</code>	This is used to return the name of the user who entered the comment

`__DDNAME.TIMESTAMP__` This is used to return the date (and time) that the comment was entered. This will be returned in the current user's date and time format that is defined in their personal settings

Three special tags are not fields in the data dictionary:

Tag name	Explanation
<code>__RESULTS__</code>	this tag returns the result string that is sent, upon execution of a call to the API. For example, if you use an HTML form to insert a record into ExtraView, and define a template that contains only the tag <code>__RESULTS__</code> , then the output would be: Problem #12342 added.
<code>__ERR_RESULTS__</code>	this tag returns any error as a result of executing the API command. It is often used in conjunction with an error handling template, as described below
<code>__RECORD_COUNT__</code>	this tag returns the number of records found from the API action named search. You may use this field in the header or footer section of a template (see below), but not in the body part of a template.

The API commands that work with templates are:

- delete
- fill_in
- get
- insert
- insert_user
- list_attachment
- search
- update
- update_user_password

The templates must be stored in a directory in your environment, typically located within your installation in a directory named `user_templates`. This directory is placed within your ExtraView installation, at the same level as the templates directory that resides inside the WEB-INF directory. This location may vary according to how you installed ExtraView. The ExtraView Administration screens have a feature that allows you to upload files directly to this directory from your local computer.

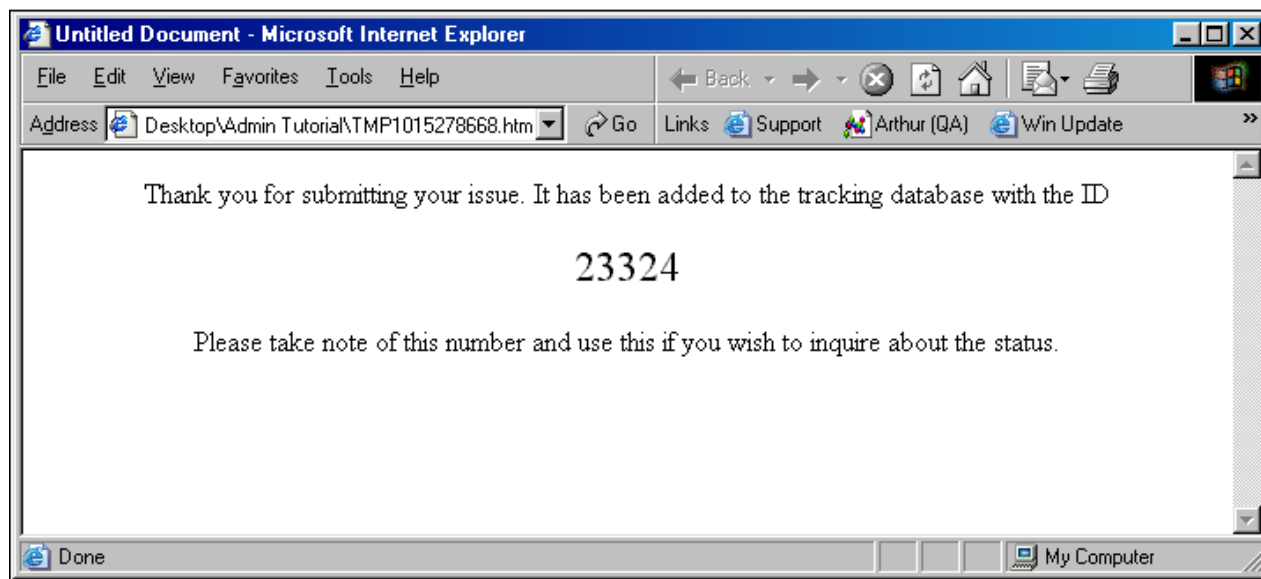
Templates are processed slightly differently, according to whether they are intended to generate text or generate HTML. First, the assumption is that if the template name has a suffix of `.html` or `.htm`, then it is assumed that it will generate HTML code. With all other file suffixes, the assumption is that they contain text. If they are HTML templates:

- The api calls *search* and *get* will have escaping enabled
- Fields that have **display_as_url** set as attribute in the data dictionary will be rendered as HTML
- Blank or null values in fields will result in ** ** being rendered

The template you define can be in one of two forms:

1. A stand-alone template that is used to simply format the results returned.

Desired Output



Template code

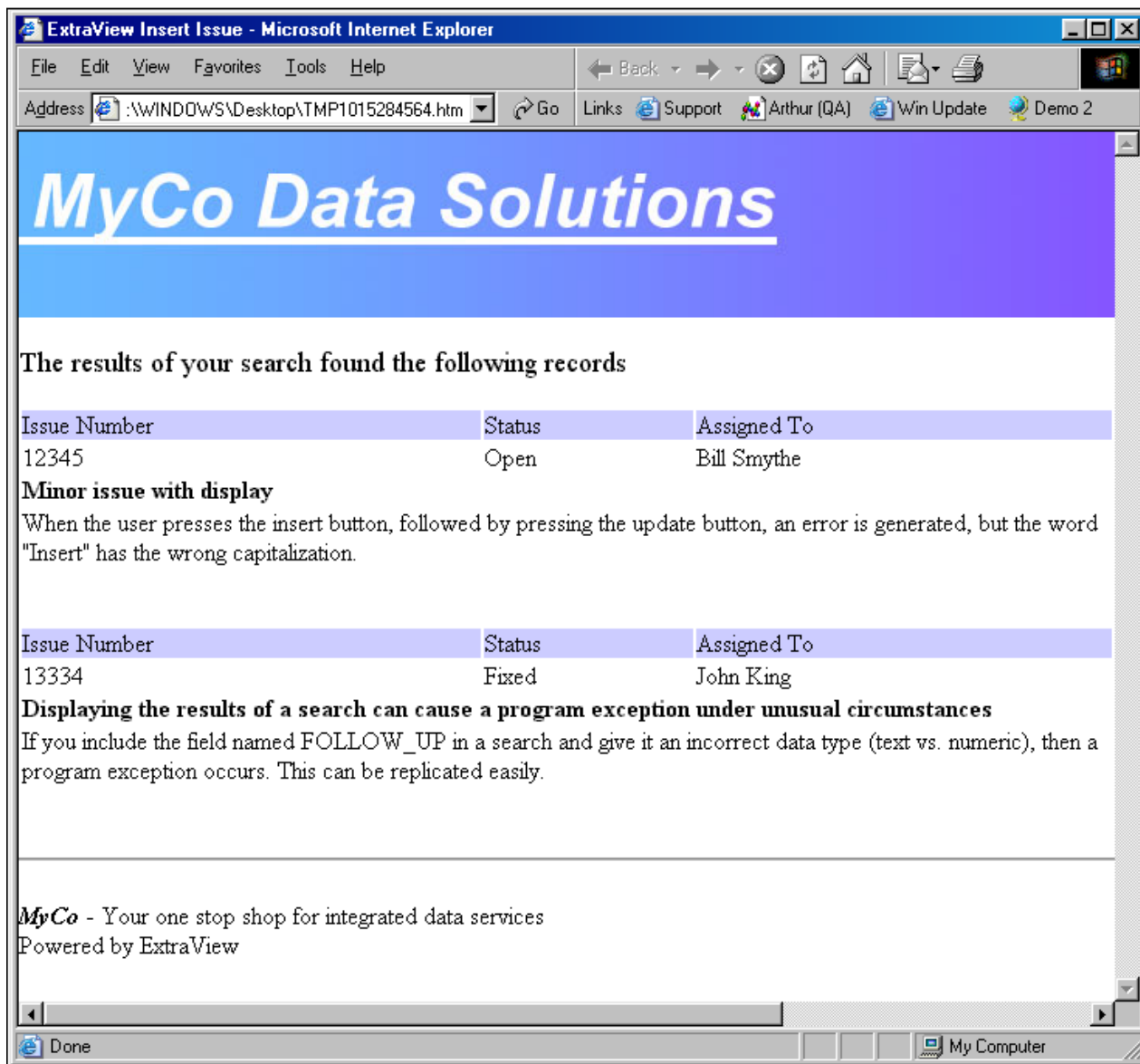
```

<html>
<head>
<title>ExtraView Insert Issue</title>
</head>
<body bgcolor="#FFFFFF">
<p align="center">Thank you for submitting your issue. It has been added
to the tracking database with the ID</p>
<p align="center"><font color="#000000" size="5">
__ID__
</font></p>
<p align="center">Please take note of this number and use this if you wish
to inquire about the status. </p>
</body>
</html>

```

2. A template structure that may have up to three sections, each of which resides in a separate file. Each file name is defined by prefixing the template name with the letter h, b or f, according to whether it is the header section, body section or the footer section. Each section is optional, although it makes no real sense to not have the body section.

Desired Output



Template code

The template code is split into three files representing the fixed header, the repeating body and the fixed footer.

Header File

```
<html>
<head>
<title>ExtraView Insert Issue</title>
</head>
<body bgcolor="#FFFFFF" leftmargin="0" topmargin="0">
<p></p>
<p><font size="4">The results of your search found the following records</font>
</p>
<table width="100%" border="0" cellpadding="0">
```

Body File

```

<tr bgcolor="#CCCCFF">
<td>Issue Number</td>
<td>Status</td>
<td>Assigned To</td>
</tr>
<tr>
<td>__ID__</td>
<td>__STATUS__</td>
<td>__ASSIGNED_TO__</td>
</tr>
<tr>
<td colspan="3"><b>__SHORT_DESCR__</b></td>
</tr>
<tr>
<td colspan="3">
__REPEAT_START__
__Description__
__REPEAT_STOP__
</td>
</tr>

```

Footer File

```

</TABLE>
<hr>
<p><i><b>MyCo</b></i> - Your one stop shop for
integrated data services<br>
Powered by ExtraView</p>
</body>
</html>

```

The fill_in Template

It is sometimes useful to be able to generate a template and populate it with values that do not originate in ExtraView's database. The fill_in action fulfills this need.

Syntax

```

http://www.myserver.com/evj/ExtraView/ev_api.action?user_id=username
&password=password&statevar=fill_in&p_template_file=this_template.html&id=12345
&any_name_at_all=Phyllis%20Mitchell ... ..

```

The template file, this_template.html, will be returned to the user's screen, with the values for id and any_name_at_all filled in.

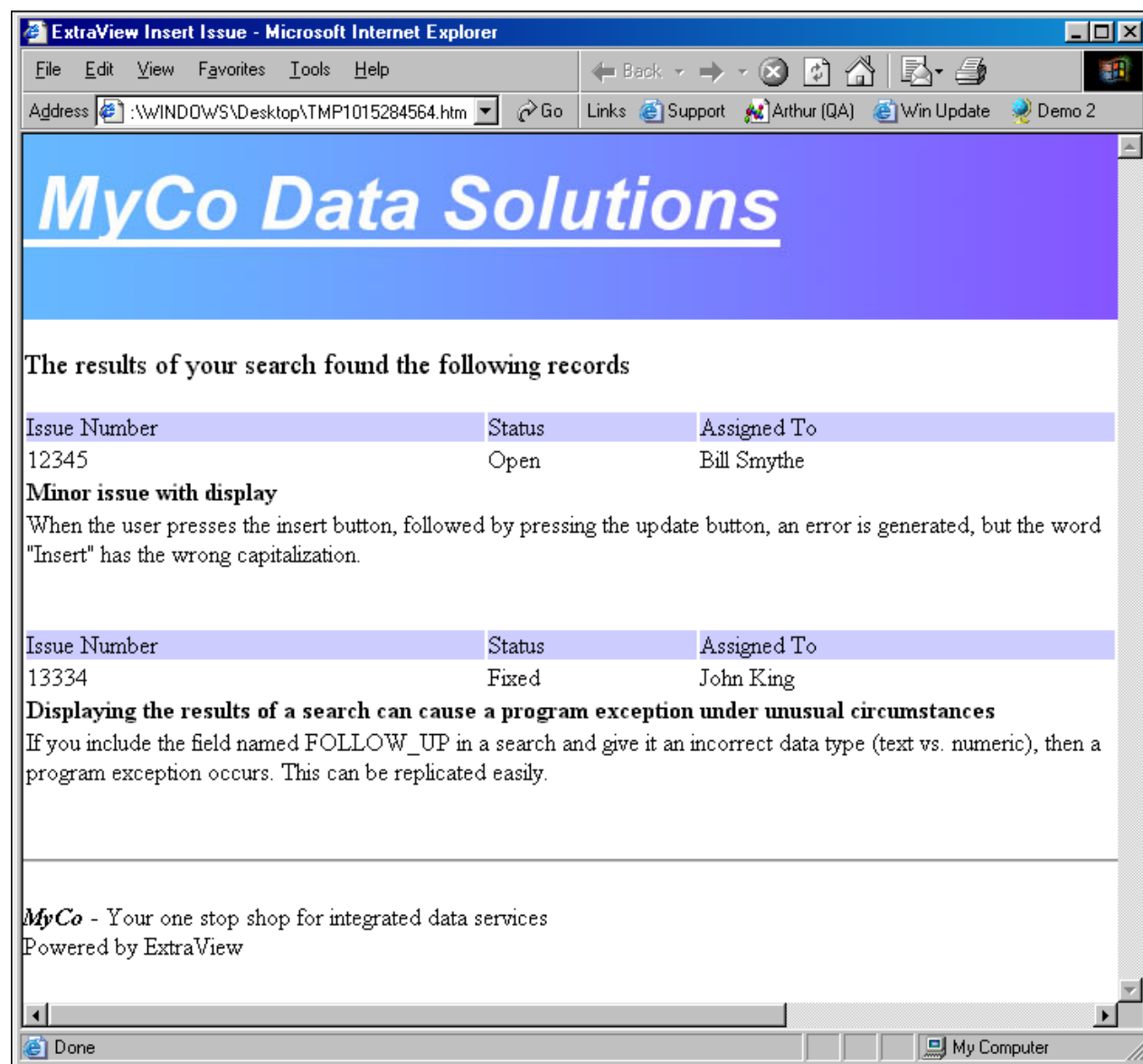
Template file

```

<html>
<head><title>ExtraView Entry Details</title></head>
<body>
<hr>
<p align=center>The ID for the problem is __ID__ and it was entered by
__ANY_NAME_AT_ALL__.</p>
<hr>
</body>
</html>

```

Browser Output



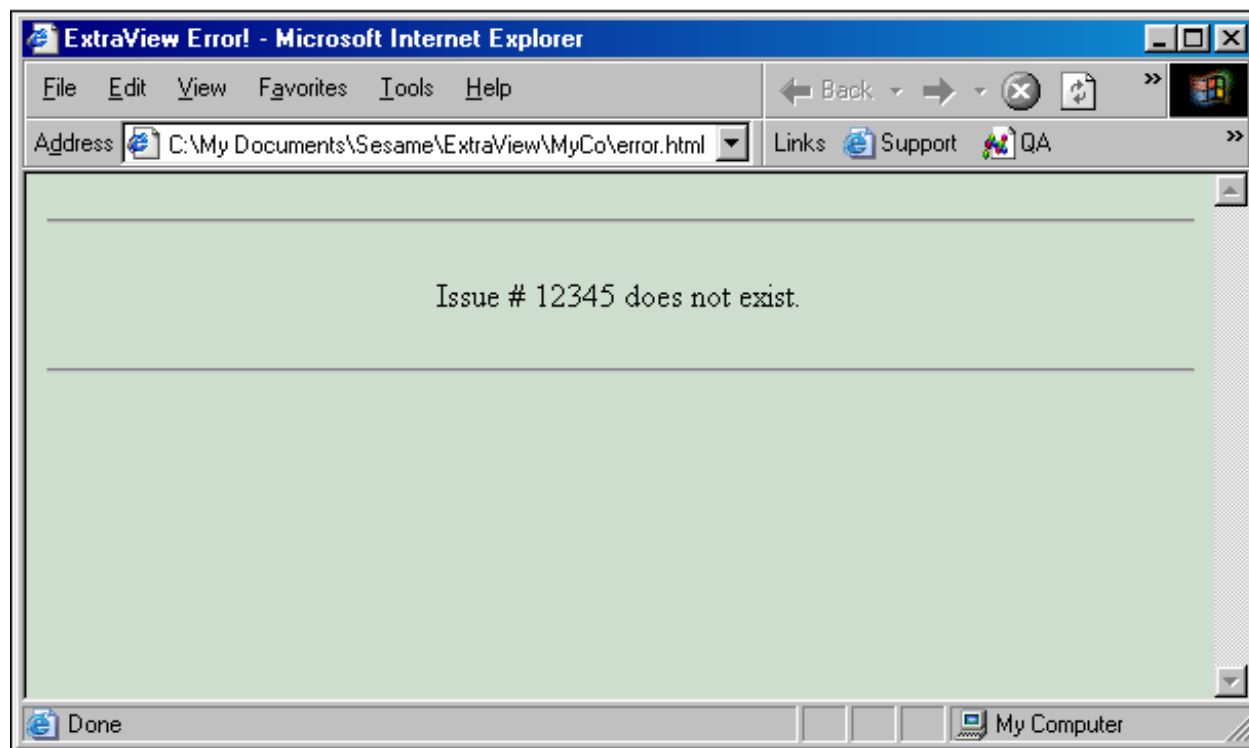
The error.html Template

If the code that executes in the API command that was submitted completes with an error or unexpected condition, the template named error.html is invoked and used to display the error to the user. Within this template, the tag `__ERR_RESULTS__` is replaced with the actual error message from the command being executed.

Template File

```
<html>
<head><title>ExtraView Error!</title></head>
<body bgcolor="#cedede">
<hr>
<p align=center>__ERR_RESULTS__</p>
<hr>
</body>
</html>
```

Browser Output



Executing ExtraView Functions

From within an ExtraView screen, you may want to add a new issue or edit an existing issue. This is typically driven by the "Display as URL" data dictionary function. This places a button on a form beside the field described in the data dictionary when you are on the add or edit screens. The "Display as URL" function can be used to open URL's both external to ExtraView and internal to ExtraView.

The reasons for doing this are various and typically related to integrating ExtraView with other enterprise applications.

It is possible to execute many different URLs within ExtraView with this technique, but some may have unpredictable results, according to the context of the request you are making. This documentation does not attempt to offer an exhaustive list of supported functions, but lists the most useful functions. Please contact ExtraView support if your requirement goes beyond this list and you have problems getting the desired functionality.

Adding an Issue

Syntax

```
http://www.myserver.com/evj/ExtraView/evSignon
?p_action=doAddDisplay
&p_option=Display
&p_close_win=true
&ev_menu=off
&p_ddname_1=value
... ..
&p_ddname_n=value
```

Notes

Use the name of your server installation in place of `http://www.myserver.com/evj/ExtraView`

If the optional parameter `p_close_win` has a value of `true`, then the *add* window will be closed when the issue is successfully updated by the user. If you set `p_close_win` to `true`, it is recommended that you also set `ev_menu` to `off`.

The optional parameter `ev_menu=off` will suppress the navigation bar within the new window that is opened to add the issue.

You may populate fields on the *add* screen that is generated, by using the convention `p_ddname=value`. Note that you should escape text fields containing special characters. For example, to add the title field to a new issue, you may use a parameter such as:

```
p_short_descr=This%20is%20the%20title%20of%20the%20issue
```

If the user is signed on when the command is issued, a new *edit* window will be opened immediately. If the user is not signed on, then the user is first taken to the sign on screen where they must sign on, before being allowed to edit the issue.

Editing an Issue

Syntax

```
http://www.myserver.com/evj/ExtraView/evSignon
?p_action=doEditDisplay
&p_option=Display
&p_id=nnnnn
&p_from_action=search
&p_from_option=search
&p_close_win=true
&ev_menu=off
```

Notes

Use the name of your server installation in place of `http://www.myserver.com/evj/ExtraView`

Replace `nnnnn` with the issue number you want to edit

If the optional parameter `p_close_win` has a value of `true`, then the *edit* window will be closed when the

issue is successfully updated by the user.

The optional parameter `ev_menu=off` will suppress the navigation bar within the new window that is opened to edit the issue.

If the user is signed on when the command is issued, a new *edit* window will be opened immediately. If the user is not signed on, then the user is first taken to the sign on screen where they must sign on, before being allowed to edit the issue.

The Email Drilldown Link

This is a special case of a drilldown used by ExtraView, providing the drilldown from within the body of an email to ExtraView. It is provided here for completeness, as it can often be used for other purposes.

Syntax

```
http://www.myserver.com/evj/ExtraView/evSignon
?p_action=doEditDisplayEmail
&p_option=Display
&p_id=nnnnn
&p_from_action=email
&p_from_option=email
&p_close_win=true
&ev_menu=off
```

Notes

Use the name of your server installation in place of `http://www.myserver.com/evj/ExtraView`.

Replace `nnnnn` with the issue number you want to edit.

If the optional parameter `p_close_win` has a value of `true`, then the *edit* window will be closed when the issue is successfully updated by the user.

The optional parameter `ev_menu=off` will suppress the navigation bar within the new window that is opened to edit the issue.

If the user is signed on when the command is issued, a new *edit* window will be opened immediately. If the user is not signed on, then the user is first taken to the sign on screen where they must sign on, before being allowed to edit the issue.

Running a Detailed Report

This function will allow you to send a URL to ExtraView, and immediately run a detailed report for an issue. If you are not signed on as a user, ExtraView will take you first to the sign on screen to authenticate your user details. You must know the id of the issue you want to display to be able to use this function.

Syntax

```
http://www.myserver.com/evj/ExtraView/evSignon
?p_option=search.SearchReportDetailDisplay
&p_action=doRunDetailed
&id=nnnnn
```


Running a Column Report

This function will allow you to send a URL to ExtraView, and immediately run an existing saved column report. If you are not signed on as a user, ExtraView will take you first to the sign on screen to authenticate your user details. You must know the report_id of the report you want to run to be able to use this function. You will need to execute the API command get_reports, to see the report_id for the report you want to run.

Syntax

```
http://www.myserver.com/evj/ExtraView/evSignon
?p_option=search.SearchReportDisplay
&p_action=doRunReport
&report_id=nn
```

Pop-up List of User Details

The purpose of this local function is to allow the administrator to program a user list or user pop-up field with the ability to pop-up a new child window, with the complete details of the user whose name is selected in the user list.

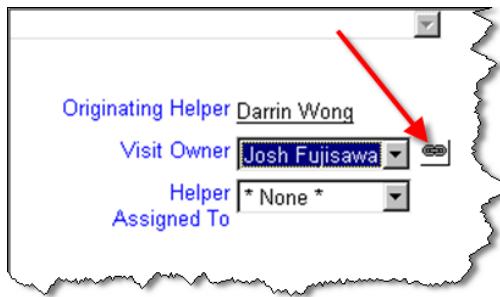
An example of how to use this function is to provide a pop-up window for the ORIGINATOR of issues within ExtraView. To achieve this, set the following text (on a single line) into the data dictionary URL field for the ORIGINATOR field:

```
?p_option=admin.UserAccountsDisplay
&p_action=showUserDetails&
p_user_id=$$NAME$$
```

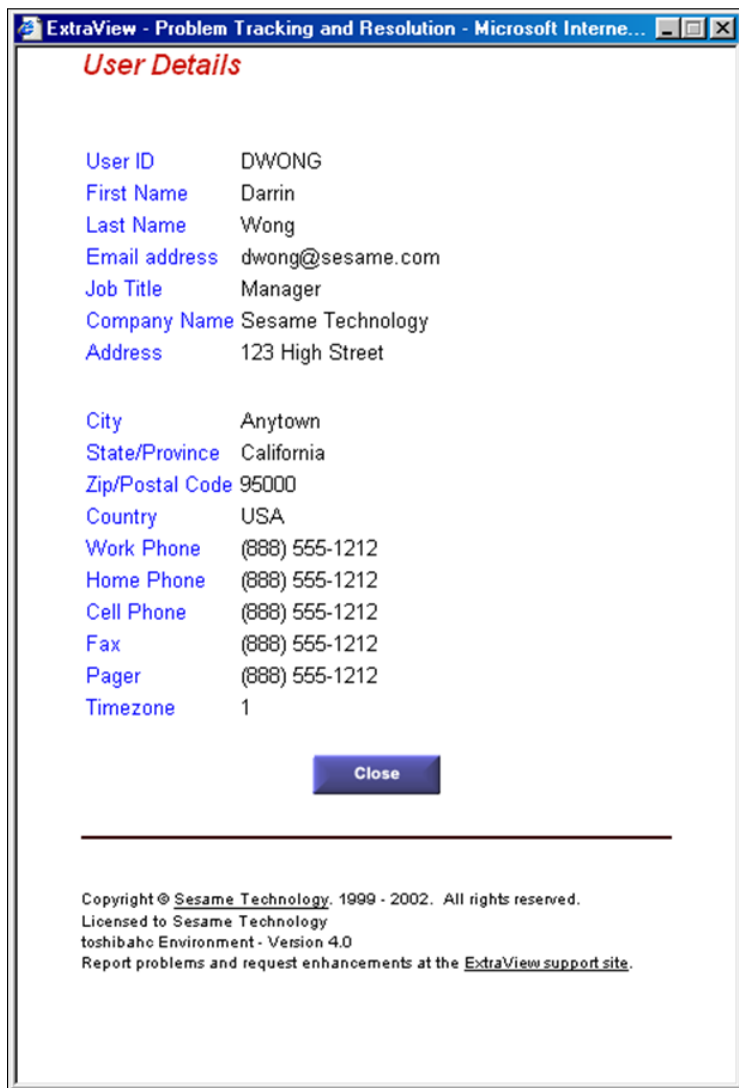
Now select Yes as the value for the field Display as URL.

Update the field in the data dictionary.

A link button will now appear by the ORIGINATOR field on both the add and edit screens as shown here.



When you press the link button, a window will pop up that shows the user's details, similar to the following.



Static Web Page Access

A common requirement is to be able to integrate ExtraView seamlessly into your own web site, allowing your own users to use a subset of ExtraView's capabilities. Most often, these users will be anonymous within ExtraView (i.e. they will not have their own user name and password), but their user details will be trapped as part of the record, for follow up.

ExtraView allows you to call it remotely, typically using a guest login, with limited privileges, to prevent "hacking" by an irresponsible user. The steps to achieving this are typically:

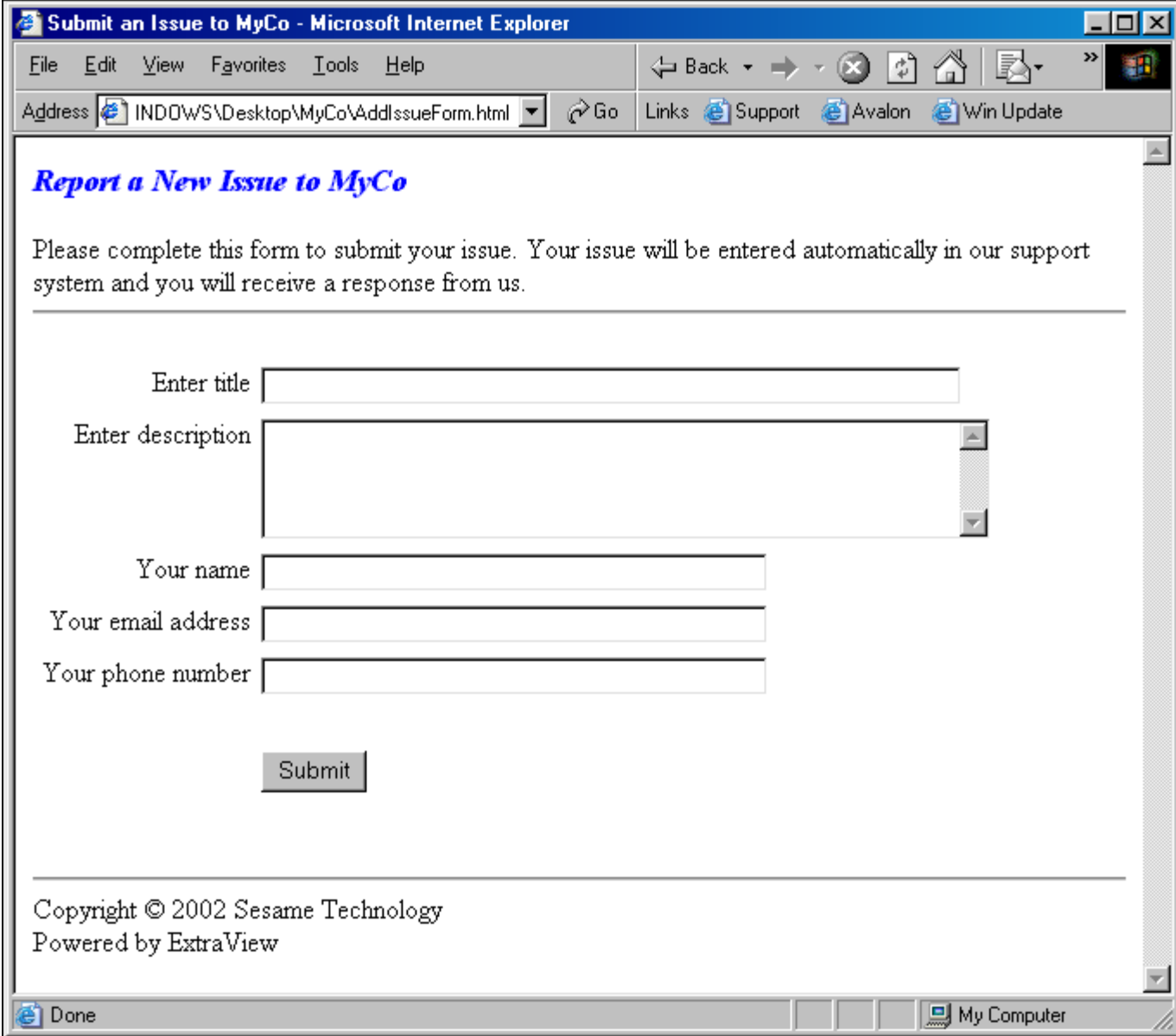
- Create or use an existing GUEST user group within the ExtraView administrative function
- Create a guest user account within ExtraView
- Use the Security Privileges to limit the fields that the GUEST user group can see and update. These will vary with the installation
- Implement HTML forms within your own website on pages. Most typically, there will be two such pages. One will allow your users to submit new issues, and one will allow users to search ExtraView for specific issues or with a keyword search
- This may also be coupled with creating specific fields within ExtraView that are used to store information that you want your users to see, as opposed to fields that are for internal use by your engineering or QA or other staff
- You may also use the privacy features of ExtraView. The ExtraView Administration Guide will help you understand this feature in depth. Issues can be either Public or Private. This means that only

users internal to your company and users within the company that reported the issue have access to the records if they are Private. If the issues are Public, then everyone can see them. You may also take a more sophisticated approach, by using Privacy Groups. These allow you to set up groups of individuals that can see specific issues, irrespective of their user group or other privileges.

HTML Pages that access ExtraView remotely

The following example shows how suitable pages can be designed. They all use the ability of ExtraView to support anonymous access, assuming the administrator has provided this facility. See the ExtraView Administrator's Guide for more information.

First, let us design a screen that allows our anonymous users to add new issues to ExtraView. It looks like this:



The screenshot shows a Microsoft Internet Explorer browser window with the title "Submit an Issue to MyCo - Microsoft Internet Explorer". The address bar contains "INDOWS\Desktop\MyCo\AddIssueForm.html". The main content area displays the following form:

Report a New Issue to MyCo

Please complete this form to submit your issue. Your issue will be entered automatically in our support system and you will receive a response from us.

Enter title

Enter description

Your name

Your email address

Your phone number

Copyright © 2002 Sesame Technology
Powered by ExtraView

The browser status bar at the bottom shows "Done" and "My Computer".

The HTML source to this page is:

```
<html>
<head>
<title>Submit an Issue to MyCo</title>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<script>
function setState(button) {
if (document.AddIssue.statevar.value != "Submitted") {
button.form.statevar.value = button.name;
}
if (button.value=="Submitted") {
alert("This form has already been submitted. Please wait.");
}
if (button.name == "Submit") {
if (document.AddIssue.short_descr.value == "") {
alert("You must enter a title for the issue");
return;
}
if (document.AddIssue.description.value == "") {
alert("You must enter a description for the issue");
return;
}
document.AddIssue.statevar.value = "INSERT";
button.value = "Submitted";
document.AddIssue.submit();
}
}
</script>
<body bgcolor="#FFFFFF">
<p><i><font color="#0000FF" size="+1"><b>Report a New Issue to MyCo</b></font>
</i></p>
<p>Please complete this form to submit your issue. Your issue will be entered
automatically in our support system and you will receive a response from us. <br>
<hr>
<br>
<form name="AddIssue" ACTION="http://myco.extraview.net/myco/ExtraView/ev_api.action"
METHOD="POST">
<INPUT NAME="statevar" TYPE="HIDDEN" VALUE="INSERT">
<INPUT NAME="status" TYPE="HIDDEN" VALUE="SUBMIT">
<INPUT NAME="assigned_to" TYPE="HIDDEN" VALUE="JIM.SMITH">
<INPUT NAME="product_name" TYPE="HIDDEN" VALUE="CUSTOMER_ISSUES">
<table border="0" cellpadding="2">
<tr>
<td width="220" valign="top">
<div align="right">Enter title</div>
</td>
<td valign="top" width="625">
<input type="text" name="short_descr" size="56" maxlength="255">
</td>
</tr>
<tr>
<td width="220" valign="top">
<div align="right">Enter description</div>
</td>
<td valign="top" width="625">
<TEXTAREA WRAP="virtual" NAME="description" COLS=50 ROWS=4></TEXTAREA>
</td>
</tr>
</tr>
</table>

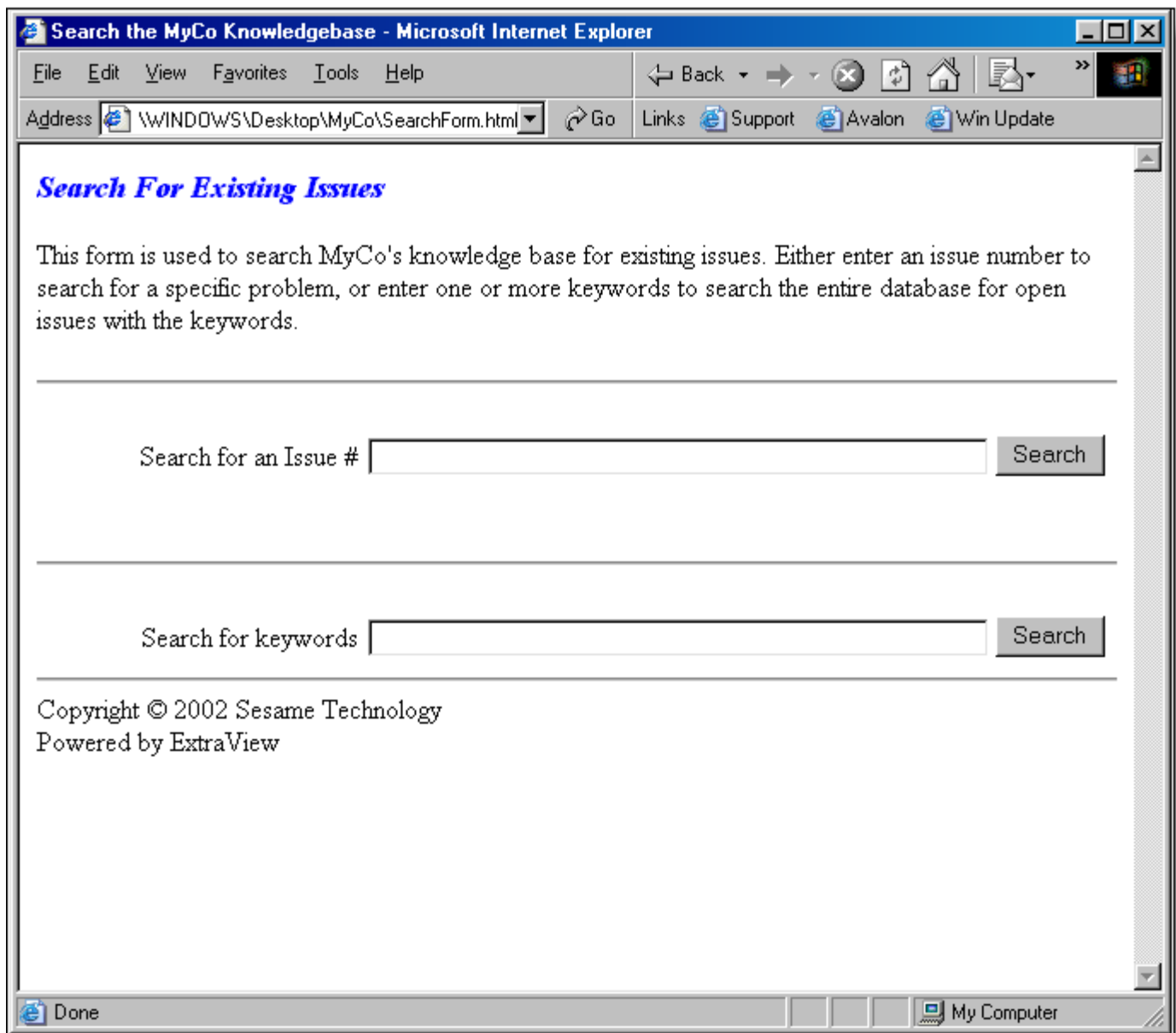
```

```

<td width="220" valign="top">
<div align="right">Your name </div>
</td>
<td valign="top" width="625">
<input type="text" name="customer_name" size="40" maxlength="40">
</td>
</tr>
<tr>
<td width="220" valign="top">
<div align="right">Your email address</div>
</td>
<td valign="top" width="625">
<input type="text" name="customer_email" size="40" maxlength="40">
</td>
</tr>
<tr>
<td width="220" valign="top">
<div align="right">Your phone number</div>
</td>
<td valign="top" width="625">
<input type="text" name="customer_phone" size="40" maxlength="40">
</td>
</tr>
<tr>
<td width="220" valign="top">
<div align="right"></div>
</td>
<td valign="top" width="625"> </td>
</tr>
<tr>
<td width="220" valign="top">
<div align="right"></div>
</td>
<td valign="top" width="625">
<input type="button" name="Submit" value="Submit" onClick="setState(this);">
</td>
</tr>
</table>
<br>
<br>
<hr>
Copyright © 2002 ExtraView Corporation<br>Powered by ExtraView
</form>
</body>
</html>

```

Next, we want to design a web page that will allow the user to search for an issue, either by the ID or by keywords. We are setting up this page to only search for OPEN issues related to a product named OUR_PROD. The page will look like this:



The source to this page is as follows:

```
<html>
<head>
<title>Search the MyCo Knowledgebase</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<script>
function search1(button) {
document.SearchForm1.submit();
}
function search2(button) {
document.SearchForm2.submit();
}
</script>
<body bgcolor="#FFFFFF">
<p><i><font color="#0000FF" size="+1"><b>Search For Existing Issues</b></font></i></p>
<p>This form is used to search MyCo's knowledge base for existing issues. Either
enter an issue number to search for a specific problem, or enter one or more
```

```

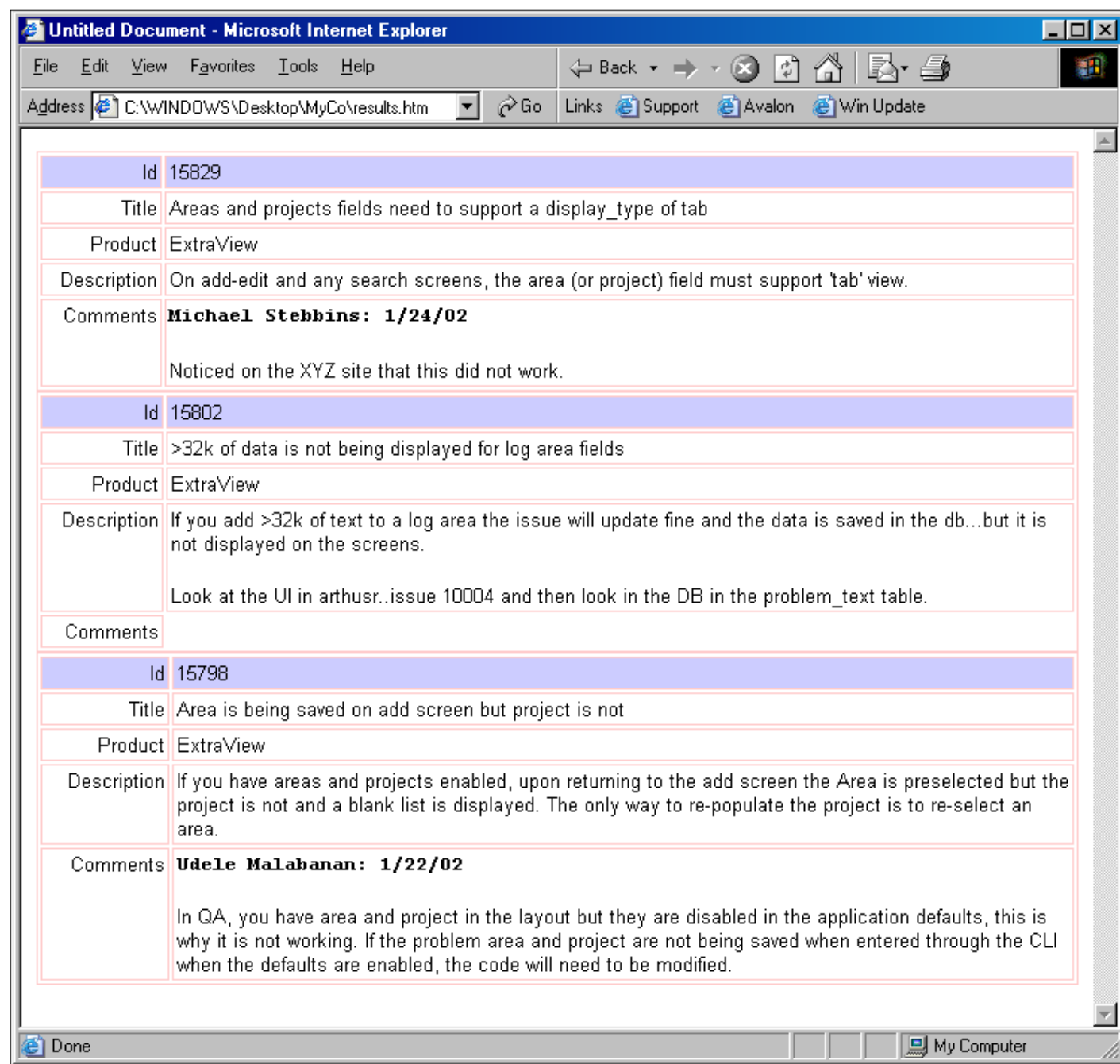
keywords to search the entire database for open issues with the keywords.<br>
</p>
<hr><br>
<form name="SearchForm1" ACTION="http://myco.extraview.net/myco/ExtraView/ev_api.action"
METHOD="POST">
<INPUT NAME="statevar" TYPE="HIDDEN" VALUE="SEARCH">
<INPUT NAME="p_PAGE_LENGTH" TYPE="HIDDEN" VALUE="10">
<INPUT NAME="p_RECORD_START" TYPE="HIDDEN" VALUE="1">
<INPUT NAME="p_TEMPLATE_FILE" TYPE="HIDDEN" VALUE="file.html">
<table width="100%" border="0" cellpadding="2">
<tr valign="middle">
<td width="30%">
<div align="right">Search for an Issue #</div>
</td>
<td>
<input type="text" name="p_id" size="50" maxlength="6">
<input type="button" name="Submit2" value="Search" onClick="search1(this)">
</td>
</tr>
</table>
</form>
<br><hr><br>
<form name="SearchForm2" ACTION=http://myco.extraview.net/myco/ExtraView/ev_api.action
METHOD="POST">
<INPUT NAME="user_id" TYPE="HIDDEN" VALUE="guest">
<INPUT NAME="password" TYPE="HIDDEN" VALUE="guest">
<INPUT NAME="statevar" TYPE="HIDDEN" VALUE="SEARCH">
<INPUT NAME="product_name" TYPE="HIDDEN" VALUE="OUR_PROD">
<INPUT NAME="status" TYPE="HIDDEN" VALUE="OPEN">
<INPUT NAME="p_PAGE_LENGTH" TYPE="HIDDEN" VALUE="100">
<INPUT NAME="p_RECORD_START" TYPE="HIDDEN" VALUE="1">
<INPUT NAME="p_TEMPLATE_FILE" TYPE="HIDDEN" VALUE="file.html">
<table width="100%" border="0" cellpadding="2">
<tr valign="middle">
<td width="30%">
<div align="right">Search for keywords</div>
</td>
<td>
<input type="text" name="p_keyword" size="50" maxlength="255">
<input type="button" name="Submit" value="Search" onClick="search2(this)">
</td>
</tr>
</table>
<hr>
Copyright © 2002 ExtraView Corporation<br>
Powered by ExtraView<br>
</form>
</body>
</html>

```

Server-side HTML Templates

In order that the search will return neatly formatted HTML in the same style as the rest of your web site, you will create a server-side ExtraView template. Note the parameter in the source file named `p_template_file` that points to the template. For example, we want to return from the search a

report that looks as follows:



The source of the template file is as follows:

```
<TABLE cellpadding="2" cellspacing="2" border="1" bordercolor="#FFCCCC">
<TR valign="top" bgcolor="#CCCCFF">
<TD align="right" width=80>Defect #</TD>
<TD width=800>__ID__</TD>
</TR>
<TR valign="top">
<TD align="right">Title</TD>
<TD>__SHORT_DESCR__</TD>
</TR>
<TR valign="top">
<TD align="right">Product</TD>
```



```

<TD>__PRODUCT_NAME__</TD>
</TR>
<TR valign="top">
<TD align="right">Description</TD>
<TD><PRE>__DESCRIPTION_TEXT__</PRE></TD>
</TR>
<TR valign="top">
<TD align="right">Comments</TD>
<TD><PRE>
<b>__COMMENTS_USER__: __COMMENTS_TIMESTAMP__</b>
<br>
__COMMENTS_TEXT__
</PRE>
</TD>
</TR>
</TABLE>

```

Note that server-side templates do not need to contain HTML. For example, if you want to output straight text for a CLI command such as `evsearch`, then a server-side template can be defined in exactly the same manner as for the HTML templates. For more information on server-side templates, please click [here](#).

Automatic Language Translation

ExtraView can be configured to translate a text area field (or similar type of field) from one language to another. This configuration can be fully automated, or may be driven by a button or other control on an *add* or *edit* screen. The feature utilizes the [Google Ajax Language API](#). At the time of writing, this API supports translations between 50 languages.

The feature has been implemented in a configurable way within ExtraView so that the administrator can tailor its implementation to the work flow needed. This page shows a typical implementation, but with a little work in JavaScript, this feature may be used in a whole variety of different ways. The example shows how to translate a text area field from English to French automatically when the field containing the English text is modified.

The ExtraView behavior setting named `ENABLE_GOOGLE_LANGUAGE_API` must be set to a value of `YES` to enable the feature. This provides the infrastructure within the *add* and *edit* screens to support the translations. Note that all users must have Internet access to reach the Google server to use this feature.

For this example, we are assuming that we will enter or modify text in the `DESCRIPTION` field, and place the translated results into a field named `TRANSLATE_RESULTS`. The logic is controlled by two JavaScript functions placed in the `UserJavaScript.js` file. Follow these steps to configure:

- Turn the behavior setting named `ENABLE_GOOGLE_LANGUAGE_API` to `YES`
- We have a function in the `UserJavaScript.js` file named `ev_translate`. The default looks like this:

```

function ev_translate(from_lang, to_lang) {
  try {
    var d = document.editForm;
    google.language.translate(d.p_description.value, from_lang, to_lang, function(result)
      {var regex = new RegExp("'", "gi");
       d.p_translate_result.value = result.translation.replace(regex, "'");});
  } catch (err) {
    // just ignore
  }
}

```

This function may be modified, for example to alter the names of the fields being referenced or to define a different structure for the two languages being referenced. For example, you might want to trigger the language for the result to be selected from a list.

- The second function in UserJavaScript.js provides the call back to the Google API. It is provided as:

```
function onloadCallback() {
  try {
    var d = document.editForm;
    google.language.translate(d.p_description.value, 'en', 'fr', function(result) {
      var regex = new RegExp("", "gi");
      if (!(typeof(d.p_description)=='undefined')
          || typeof(d.p_translate_result=='undefined')) {
        d.p_translate_result.value = result.translation.replace(regex, "");
      }
    })
  } catch (err) {
    // just ignore
  }
}
```

Again, significant modification may be made to this function. The example here simply translates the text in the DESCRIPTION field from English to French and places the result in the TRANSLATE_RESULT field.

- Make sure you place the DESCRIPTION and the TRANSLATE_RESULT fields on the layouts where they are to be used and make sure that they have read and write permission
- To trigger the translation, you simply create an HTML modifier in the DESCRIPTION field within the *add* or *edit* screens where you want to use the feature. This example simply looks for a change in the content to trigger the translation:

```
onchange=ev_translate('en', 'fr');
```

add_attachment

This action uploads a file from the local file system to be attached to an existing record in ExtraView. It is designed to be used within an HTML form.

Note that the syntax and usage of this command is different to other API commands; extra care should therefore be taken when using the `add_attachment` command.

SYNTAX

```
<FORM METHOD="post"
action=http://www.myserver.com/dev/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=add_attachment
&p_template_file=template_filename
>
```

This command must have additional parameters that are not part of the FORM tag, but are part of the HTML within the `<form> </form>` construct. These will be provided as part of the INPUT tags within the form. The INPUT tags that must be provided are:

Tag name	Purpose
<code>p_id</code>	The issue ID
<code>p_attach_desc</code>	The description of the attachment

file The filename of the attachment

The following example shows how this API command is used from within an HTML form. The example includes the use of a template file that formats the results returned from ExtraView.

Adding an attachment from an HTML page

```
<html>
<title>Add an attachment to an existing issue</title>
<body>
<form method="POST"
action="http://myserver.extraview.net/dev/ExtraView/ev_api.action?statevar=add_attachment
&p_template_file=attach_results.html&p_id=12345" enctype="multipart/form-data">
<input type="hidden" name="p_id" value="12345">
<table>
<tr>
<td>Description</td>
<td><input type="text" size="40" name="p_attach_desc"></td>
</tr>

<tr>
<td colspan="2">Add attachment</td>
</tr>

<tr>
<td>Filename</td>
<td><input type="file" size="40" name="file"></td>
</tr>

<tr>
<td colspan="2">
<input type="submit" value="Add Attachment">
</td>
</tr>
</table>
</form>
</body>
</html>
```

Note that if you are using a template file to return the results of the command, you must include the parameter named `p_id` twice within the HTML, once within the `<form ...>` tag, and once as a hidden field within the `<INPUT ...>` tag. This is because of the limitations of how browsers work with multi-part forms.

attach_results.html template file

This file must reside in the `WEB-INF/user_templates` directory. Please see the page on [server-side templates](#) for a fuller explanation. Carefully note the following in the example file. The following fields are available as tags within the template.

You will see in the example that there is a `__REPEAT_START__`, `__REPEAT_STOP__` block. This allows the display of all attachments added to the issue, with the one just added being the first in the list. If you only want to see the details of the attachment you just added, you can remove the `__REPEAT_START__` and `__REPEAT_STOP__` statements.

Tag name	Purpose
<code>__ATTACH_DESC__</code>	The description of the file that was attached
<code>__ATTACHMENT_ID__</code>	The internal ID of the attachment. Typically not useful for external purposes
<code>__CREATED_BY_USER__</code>	The user who added the attachment
<code>__DATE_CREATED__</code>	The date the attachment was created
<code>__ID__</code>	The ID of the issue to which the attachment was added
<code>__FILE_NAME__</code>	The filename of the file attached
<code>__FILE_SIZE__</code>	The size, in bytes, of the attached file
<code>__NFILES__</code>	The total number of files attached to the issue

```
<html>
<title>
Attachment added to ExtraView
</title>
<body>
<p>Attachment added successfully to issue # __ID__</p>
<br>
A list of all attachments held in the issue is as follows
<br><br>
<table>
<tr>
<td>File name</td>
<td>Description</td>
<td>File size</td>
<td>Created By</td>
<td>Date Created</td>
</tr>
__REPEAT_START__
<tr>
<td>__FILE_NAME__</td>
<td>__ATTACH_DESC__</td>
<td>__FILE_SIZE__</td>
<td>__CREATED_BY_USER__</td>
<td>__DATE_CREATED__</td>
</tr>
__REPEAT_STOP__
</table>
<br>
</body>
</html>
```

NOTES

- This command uses a different syntax to most other commands within the API. This is to allow ExtraView to handle the multi-part form, used for uploading files to the server
- You can upload and attach the same file multiple times to a single record within ExtraView
- There are no limits to the number of files that can be attached to a single record
- Each file uploaded can be up to 4 GB in size
- The syntax of this command is likely to change with future versions of ExtraView, to be consistent with the same action structure as other commands

add_field_list

This action provides a list of all the available fields to the user in a specific order that is used by the layout for adding new issues within ExtraView. Most typically, this script is used to provide a list of fields and their titles for inclusion in a Perl script that is used to insert a new record within the ExtraView database. This action is used as a basis of the CLI evadd command.

Note that all of the ExtraView security is in force and an individual user will only see the fields to which he has access. Also, note that there is no difference in the way that User Defined Fields (UDF's) are shown from other fields. UDF's are handled in a seamless way within the API.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=add_field_list
&include_images=1
```

This will return a list of fields as shown in the following figure. Note that the fields will vary according to your permissions and the fields defined in your installation.

```
RESOLUTIONDisposition
CUSTOMERCustomer
+SHORT_DESCRTitle
ALT_IDAlt ID
+PRODUCT_NAMEProduct
PRIORITYPriority
+CATEGORYCategory
SEVERITY_LEVELSeverity
ASSIGNED_TOAssigned To
OWNEROwner
COMPONENTComponent
TEST_CASE_IDTest Case ID
TEST_CASE_LOCATIONTest Case Location
PLATFORMPlatforms
OSOS
PRIVACYView
*WORKAROUNDWorkaround
*RELEASE_NotesRelease Notes
+*DescriptionDescription
```

NOTES

The general form of each entry returned by the command is:

```
<prefix><fieldName><delimiter><fieldTitle><delimiter><parentName>
```

where the `parentName` is blank or the immediate parent in an allowed-value or database-type relationship and the `delimiter` is that specified in the behavior setting named `DEFAULT_TEXT_REPORT_DELIMITER` and the characters in the prefix have the following meaning:

~ means the field is part of a repeating row layout
 * means the field has a display type of `textarea` or `logarea`
 % means the field has a display type of `user`

- The parameter `include_images` is optional. If provided, it always has a value of 1. When the parameter is provided, fields with a display type of `image` are included in the results returned.

add_udf_list

This command inserts new list values into existing user defined list type fields.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=add_udf_list
&udf_name=my_field
&udf_values=val
```

To add more than one list value with a single API command, you need to have encoded the `tab` character separator in your API command.

EXAMPLE

This example adds `xxx`, `yyy` and `zzz` as values to a UDF named `my_field`.

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&STATEVAR=add_udf_list
&UDF_NAME=my_field
&UDF_VALUES=xxx%09yyy%09zzz
```

add_user_to_group

This action adds an existing user to an existing user group.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=add_user_to_group
```

```
&security_user_id=userID  
&user_group=userGroup
```

NOTES

- You must provide both an existing user and an existing user group
- The command will fail if you pass a name other than security_user_id or user_group
- The command will fail if the user is already a member of the user group
- You must have update permission to the security key named SE_SECURITY_GROUP before you can execute the command

allowed_list

This command retrieves a list of allowed values for a given parent key. For example, if modules (MODULE_ID) in your installation are dependent upon products (PRODUCT_NAME), then you can use this function to find all the valid modules for a given product. The command can also be used to retrieve the values in a specified list only.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=allowed_list  
&field=field_name  
&parent=parent_field_name  
&parent_val=product_name
```

NOTES

Both the field_name and the product_name must exist else the command will fail.

The delimiter of a colon (:) in the example is the system delimiter stored in the behavior setting named DEFAULT_TEXT_REPORT_DELIMITER.

If you only specify the field without the parent and parent_val, the command will return a list of all the values of the field.

EXAMPLE

The following URL retrieves a list of modules for the product named WIDGET.

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=allowed_list  
&field=MODULE_ID  
&parent=PRODUCT_NAME  
&parent_val=WIDGET
```

This will produce output similar to the following:

```
1022:POWER_SUPPLY
1032:CASE
1203:MOTHERBOARD
1255:KEYBOARD
1334:MOUSE
1432:MONITOR
```

custom

This call invokes the CLI user exit in the UserCustom Java class.

SYNTAX

```
http://www.myserver.com/evj/extraView/ev_api.action?
user_id=username
&password=password
&statevar=custom
[&custom_param1=value[&custom_param1=value...]]
```

NOTES

This command works in conjunction with the UserCustom Java class within ExtraView. This command can be used to create your own API calls that execute any code within the ExtraView environment. It is therefore a very powerful command that can be used to create any new API command or commands that you need for any purpose.

The custom parameters are optional and may be any name value pairs that you provide to the new command.

The parameters are passed into the UserCustom CLI exit.

debug

This command sets or resets the internal debug level of messages being sent to ExtraView's log file.

SYNTAX

```
http://www.myserver.com/evj/ExtraView?DEBUG=nn
```

NOTES

The default level for nn is 6. Valid values are in the range 1 through 12.

This command affects all users of ExtraView, no matter how they access the program (CLI, Web interface, API), and the higher the value, the more the performance of ExtraView is degraded for all users. In addition, considerable more information is written to the log with higher values of the debug

level. Therefore, please ensure the level is set back to a maximum of 6 in your production environment, if you have altered it while testing scripts that you are developing.

delete

This action allows you to delete an existing record within ExtraView's database. Note that you must have permission to delete records before you can execute this action. The security key that controls this is named PR_RESOLUTION.DELETE_BUTTON.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=delete  
&p_template_file=file.html  
&id=nnnnn
```

If the issue is deleted without error, the function will return the issue number as shown below. Note that the term "Bug #" is dependent on the screen title used for the ID field in the data dictionary.

```
Bug # 12352 deleted.
```

NOTES

You must provide a valid name and value pair for the ID field. The value must be an existing issue within the ExtraView database. If you do not provide this, an error message is generated.

delete_user

This action deactivates an existing user from ExtraView. **It does not delete the user record from the database.** This is because historic records contain references to users and their name must remain available for display.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=delete_user  
&security_user_id=userID
```

NOTES

You must provide an existing userID.

The command will fail if you pass a name other than security_user_id.

You must have update permission to the security key named SE_SECURITY_USER before you can execute the command.

edit_field_list

This action provides a list of all the available fields to the user in a specific order that is used by the layout for updating existing issues within ExtraView. Most typically this script is used to provide a list of fields and their titles for inclusion in a script that is used to update an existing record within ExtraView's database. This action is used as a basis of the CLI command `evupdate`.

Note that all of ExtraView's security is in force and an individual user will only see the fields to which he has access. In addition, there is no difference in the way that User Defined Fields (UDF's) are shown compared to other fields. UDF's are handled in a seamless way within the API.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=edit_field_list
&include_images=1
```

This will return a list of fields as shown in the following figure. Note that the fields will vary according to your permissions and the fields defined in your installation.

```
RESOLUTIONDisposition
CUSTOMERCustomer
+SHORT_DESCRTitle
ALT_IDAlt ID
+PRODUCT_NAMEProduct
PRIORITYPriority
+CATEGORYCategory
SEVERITY_LEVELSeverity
ASSIGNED_TOAssigned To
OWNEROwner
COMPONENTComponent
TEST_CASE_IDTest Case ID
TEST_CASE_LOCATIONTest Case Location
PLATFORMPlatforms
OSOS
PRIVACYView
*RELEASE_NotesRelease Notes
+*DescriptionDescription
```

NOTES

The general form of each entry returned by the command is:

```
<prefix><fieldName><delimiter><fieldTitle><delimiter><parentName>
```

where the `parentName` is blank or the immediate parent in an allowed-value or database-type relationship and the `delimiter` is that specified in the behavior setting named `DEFAULT_TEXT_REPORT_DELIMITER` and the characters in the prefix have the following meaning:

~ means the field is part of a repeating row layout

- * means the field has a display type of textarea or logarea
- % means the field has a display type of user

The parameter `include_images` is optional. If provided, it always has a value of 1. When the parameter is provided, fields with a display type of image are included in the results returned.

fields

This action provides a list of all the available fields to the user. Note that all of ExtraView's security is in force and an individual user will only see the fields to which he has access. Also note that there is no difference in the way that User Defined Fields (UDF's) are shown than other fields. UDF's are handled in a seamless way within the API.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=fields
&include_fields=y
```

NOTES

The `include_fields=y` is an optional name/value pair. If this is included then the output returned returned by the command will have the display type of each field returned, following the title of the field.

This will return a list of fields similar to that shown in the following figure. Note that the fields will vary according to your permissions and the fields defined in your installation.

```
ASSIGNED_TOAssigned To
CATEGORYCategory
COMPONENTComponent
DATE_CLOSEDDate Closed
DATE_CREATEDCreated
DAYS_IN_STATUSDays in Queue
DAYS_OPENDays Open
IDBug #
MONTHS_IN_STATUSMonths In Status
MONTHS_OPENMonths Open
ORIGINATOROriginator
ORIGINATOR_NAMEOriginator
OSOS
OWNEROwner
PLATFORMPlatforms
PRIORITYPriority
PRIVACYView
PRODUCT_NAMEProduct
RELEASE_FIXEDVersion Closed
RELEASE_FOUNDVersion Open
*RELEASE_NotesRelease Notes
RELEASE_STATUSRelease Status
RESOLUTIONDisposition
SEVERITY_LEVELSeverity
SHORT_DESCRTitle
```

TIMESTAMPLast Modified
 TIMESTAMP_MONTHTimestamp Month
 TIMESTAMP_WEEKTimestamp Week
 WEEKS_IN_STATUSWeeks In Status
 WEEKS_OPENWeeks Open
 *WORKAROUNDWorkaround

get

This API call retrieves an individual record from the ExtraView database. You must know the ID of the issue in question to be able to extract the information.

SYNTAX

```

http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get
&id=nnnnn
&p_template_file=file.html
&username_display=ID | LAST | FIRST
  
```

The action retrieves a single record in XML format from the database and displays it similarly to the example shown in the following figure. Note that only fields to which the user has permission will be displayed. Also, note that it is possible to see repeating fields such as version records within the output.

```

<?xml version="1.0"?>
<PROBLEM_RECORD>
<ID TITLE="Bug #">12621</ID>
<SHORT_DESCR TITLE="Title"><![CDATA[An error occurs when you overload the
power convertor]]></SHORT_DESCR>
<SEVERITY_LEVEL TITLE="Severity">High</SEVERITY_LEVEL>
<PRIORITY TITLE="Priority">Low</PRIORITY>
<PRODUCT_NAME TITLE="Product">NetPower</PRODUCT_NAME>
<DATE_CREATED TITLE="Created">14-AUG-01</DATE_CREATED>
<OWNER TITLE="Owner">rick</OWNER>
<TIMESTAMP TITLE="Last Modified">15-AUG-01</TIMESTAMP>
<ASSIGNED_TO TITLE="Assigned To">Florence</ASSIGNED_TO>
<PRIVACY TITLE="View">Private</PRIVACY>
<CATEGORY TITLE="Category">Hardware</CATEGORY>
<RESOLUTION TITLE="Disposition">Not found</RESOLUTION>
<DATE_CLOSED TITLE="Date Closed"></DATE_CLOSED>
<ORIGINATOR TITLE="Originator">JON.BJORNSTAD</ORIGINATOR>
<ORIGINATOR_NAME TITLE="Originator">Jon Bjornstad</ORIGINATOR_NAME>
<RELEASE_RECORD>
<RELEASE_FOUND TITLE="Version Open">ADC2</RELEASE_FOUND>
<RELEASE_FIXED TITLE="Version Closed">1.01</RELEASE_FIXED>
<RELEASE_STATUS TITLE="Release Status">Unassigned</RELEASE_STATUS>
</RELEASE_RECORD>
<RELEASE_RECORD>
<RELEASE_FOUND TITLE="Version Open">Framework 1.3</RELEASE_FOUND>
<RELEASE_FIXED TITLE="Version Closed">Framework 1.3</RELEASE_FIXED>
  
```

```
<RELEASE_STATUS TITLE="Release Status">Open</RELEASE_STATUS>
</RELEASE_RECORD>
<Description TITLE="Description"><![CDATA[If you enter an overload trip on the
front panel you will find an error occurs.]]></Description>
</PROBLEM_RECORD>
```

NOTES

You must provide a valid name and value pair for the ID field. The value must be an existing issue within the ExtraView database. If you do not provide this, an error message is generated.

The optional parameter USERNAME_DISPLAY may be used to override the behavior setting named USERNAME_DISPLAY, for the duration of the execution of a single API call. This allows the developer to return the user names in a different format than the system-wide default.

The fields returned in the results correspond to the fields on the detailed report layout of the user's current business area, current project and current role. However, if the behavior setting named REPORT_DTL_ITEM_DATA_LAYOUT is set to YES, then the detailed report for the user's current role and the issue's business area and project are used to define the fields being returned.

get_areas

This action retrieves a list of areas that exist within the ExtraView database. For a full explanation of areas, please consult the [Administration Guide](#).

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_areas
```

NOTES

The delimiter of a colon (:) in the example is the system delimiter stored in the behavior setting named DEFAULT_TEXT_REPORT_DELIMITER.

The currently selected area for the user who is executing the command, in their current role is marked with an asterisk (*).

Sample return from the get_areas function:

```
*0:(default values)
3:Customer
23:Project
43:Incident
```

get_attachment

This action downloads a file attached to an existing record in ExtraView to the local file system.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_attachment
&attachment_id=nnnnn
```

NOTES

You can retrieve the filenames and attachment ID's with the command list_attachment.

An example of this command is:

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=myuser&password=mypassword&statevar=list_attachment&id=10070
```

Generates this result:

```
2010-6-24.9.49.:UserJavaScript.js:3639:ExtraView:23:test:application/x-javascript
```

get_behavior_setting

This action retrieves the value of a behavior setting from ExtraView.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_behavior_setting
&app_default_name=1
```

NOTES

You must provide the name of an existing behavior setting in the get_behavior_setting parameter name.

The result of the function is:

```
app_default_nameapp_default_value
```

where the delimiter " is the system delimiter stored in the behavior setting named DEFAULT_TEXT_REPORT_DELIMITER.

The =1 in the parameter list is a mechanism to provide a dummy value. The number 1 has no significance.

For a full list of behavior settings and their uses, please consult the [Administration Guide](#).

get_field_defaults

This action returns the data dictionary defaults for all data dictionary fields that have a default value set.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_field_defaults
```

NOTES

The information returned from this call is of the format:

```
field_name:default_value
```

One row of data is returned for each field in the data dictionary that has a default value.

get_fields

This API call retrieves specific fields from an individual record from the ExtraView database. You must know the ID of the issue in question to be able to extract the information.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_fields
&username_display=ID | LAST | FIRST
&id=nnnnn
&status=1
&priority=1
&short_descr=1
```

The action retrieves only the fields requested from a single issue record in the database and displays the results with one field on each line. Note that only fields to which the user has permission will be displayed. Also, note that it is possible to see repeating fields such as version records within the output.

The above command with the syntax given will return output similar to:

```
Configuring the XYZ module results in an error
OPEN
MEDIUM
```

NOTES

You must provide a valid name and value pair for the ID field. The value must be an existing issue within the ExtraView database. If you do not provide this, an error message is generated.

The values returned may not be returned in the same order as the parameters you provide.

The values returned are the NAMES or ID's of the fields set as parameters.

The =1 in the parameter list is a mechanism to provide a dummy value. The number 1 has no significance.

The optional parameter `username_display` may be used to override the behavior setting named `USERNAME_DISPLAY`, for the duration of the execution of a single API call. This allows the developer to return the user names in a different format than the system-wide default.

get_heartbeat

This API call provides an indication of the status of ExtraView.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_heartbeat
```

NOTES

The return from the server is XML, similar to the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<EV_HEARTBEAT>
<EV_STATUS>EXTRAVIEW ALIVE</EV_STATUS>
<DB_STATUS>DB CONNECTION CONFIRMED</DB_STATUS>
<DB_DATETIME>2003/09/11 11:33:22</DB_DATETIME>
<FREE_MEMORY>197</FREE_MEMORY>
<TOTAL_MEMORY>250</TOTAL_MEMORY>
<HEARTBEAT_EXEC_TIME>330</HEARTBEAT_EXEC_TIME>
<TASK_INFO TASK=task name>
<NODE_ID>node name</NODE_ID>
<START_OPTION>start option</START_OPTION>
<TASK_STATE>task state</TASK_STATE>
<POLL_INTERVAL>poll interval seconds</POLL_INTERVAL>
- [<THREAD_INFO>
<THREAD_STATE>thread state</THREAD_STATE>
<SECS_SINCE_EXECUTION>seconds since last execution
[<PRIORITY>thread priority</PRIORITY> ]
</THREAD_INFO>]
</TASK_INFO>
</EV_HEARTBEAT>
```


In the above, the section on task information is repeated for each configured task on the application server. The returned information includes:

task name	name of the background task, e.g., SESSION_MONITOR, TASK_CONTROL_TASK, BATCHMAIL, etc.
node name	name of the node hosting the evapi service
start option	START_NOW, STOP_NOW, START_ON_BOOT, or none
task state	STARTED, STOPPED, or ERROR
poll interval seconds	the (minimum) number of seconds between polled executions
thread state	running or stopped
seconds since last execution	number of seconds since the task was most recently scheduled to run
thread priority	the priority of the thread (using Java thread priority values) -- may not appear in output

The return indicates not only that ExtraView is alive, but also it confirms that a database connection could be made, that the ExtraView servlet is running on the application server, and that the web server is running.

Note that the tag named DB_DATETIME and its value provide the current timestamp of the database server. This can be useful to provide local client applications with the server time of the host ExtraView application.

The amount of free memory, the total memory and the amount of time the command took to execute are also returned.

This command can be placed in a script that is run at routine intervals to provide an indication of the health of the system. Not only can confirmation be made that the system is alive, but the time to execute the command is available

get_log

This command returns the contents of the ExtraView application server log.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_log
```

NOTES

ExtraView makes a check to ensure that the user requesting the log has administrative privileges, as indicated by the behavior setting named ADMIN_BYPASS_GROUP being one of the user roles that the user may adopt. If the user does not have this access, they will not be able to execute this command successfully.

The output from this command may be substantial in size.

get_projects

This action retrieves a list of projects that exist within an area in the ExtraView database. For a full explanation of areas and projects, please consult the [Administration Guide](#).

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=get_projects  
&area_id=nn
```

NOTES

The value of the area_id, nn, must exist in the database, else the command will fail.

The asterisk (*) in the return shows the currently selected project.

The delimiter ':' in the example is the system delimiter stored in the application default named DEFAULT_TEXT_REPORT_DELIMITER.

Sample return from the get_projects function:

```
*0:(default values)  
1:Customer Projects  
2:Internal Projects  
3:Documentation  
4:Marketing Requirements
```

get_reports

This function retrieves a list of available reports for a given user.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=get_reports
```

NOTES

The delimiter ':' in the example is the system delimiter stored in the behavior setting named `DEFAULT_TEXT_REPORT_DELIMITER`.

The typical use of this command is to retrieve a list of available public and private reports, to present these as a menu and allow the user to select which report he is going to run with the `run_report` function.

There are two main sections returned, a list of private: and a list of public: reports. Within each of these sections, each report is shown with four values, separated by the `DEFAULT_TEXT_REPORT_DELIMITER` and specified as:

1. Report ID – a numeric identifier for the report
2. Title – the title of the report
3. Type – One of the following types:

Type	Purpose
QUICKLIST	QuickList reports
DETAILED	Detailed reports
STANDARD	Column reports with standard filters
ADVANCED	Column reports with advanced filters
SUMMARY	Summary reports with standard filters
SUMMARY_ADV	Summary reports with advanced filters
CHART	Charts with standard filters
CHART_ADV	Charts with advanced filters
AGING	Aging reports with standard filters
AGING_ADV	Aging reports with advanced filters

Sample return from the `get_reports` function:

private:

169:All Features Requests:STANDARD:That are open
 94:Bugs I Fixed in January:STANDARD:Bugs not Closed Yet
 99:Bugs Open By Month:CHART:For Tracker Enterprise
 401:Build 30:STANDARD:Estimated versus Actual Time
 89:My Hot Llist:STANDARD:Priority 1 issues that are not closed
 73:Open and Fixed Defects:CHART:Year to Date
 114:Owners of open bugs:STANDARD:Sorted by owner
 119:Report of Open issues:STANDARD:By Bill
 109:Status of Bugs Reported:CHART:Over all bugs

public:

144:Bug Trend Report:CHART:Open vs. Fixed Issues
 391:Bugs:AGING:Aging of all bugs that are not closed
 4:Bugs - Assigned to you:SUMMARY:Issues assigned by product
 164:Bugs - List of Closed Issues:STANDARD:Ordered by Product
 68:Bugs - No Reproducible State:STANDARD:Open Issues
 386:Customer Issues:AGING:Aging of all issues
 43:Customer Issues:STANDARD:New Issues By Priority

48:Customer Issues:STANDARD:Open Issues By Priority
249:Customer Issues - Time Spent:STANDARD:Items fixed
58:Feature Requests:SUMMARY: Requests By Product and Category
239:Helpdesk Issues:STANDARD:New / Open Issues Assigned to Me
234:Helpdesk Issues:ADVANCED:Open Issues
209:Knowledge Base:ADVANCED:Published Articles
214:Knowledge Base:ADVANCED:Unpublished Articles
79:My Open Issues:STANDARD:for Home Page
33:Open P1 Issues:STANDARD:All Areas - Ordered by Assigned To
3:Originated by you:SUMMARY:Issues originated by you
63:QA List - Fixed Defects:STANDARD:Ordered by Priority
84:Summary of All Issues:CHART:Chart of All Statuses
53:Summary of Open Customer Issues:SUMMARY_ADV:By Product

get_roles

This action retrieves a list of available roles for a given user.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=get_roles
```

NOTES

The delimiter ':' in the example is the system delimiter stored in the application default named DEFAULT_TEXT_REPORT_DELIMITER.

The asterisk (*) indicates the current role of the user.

Sample return from the get_roles function:

```
ENGINEERING:Engineering  
*QA:Quality Assurance  
MNGMT:Management  
ADMIN:Administrator
```

get_title

This action retrieves the title of a field from the ExtraView data dictionary, by providing its field name.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password
```

```
&statevar=get_title
&dd_name=1
```

NOTES

You must provide the name of an existing database dictionary field in `dd_name`.

The `=1` in the parameter list is a mechanism to provide a dummy value. The number 1 has no significance.

The result of the function is:

```
dd_nametitle
```

where the delimiter `"` is the system delimiter stored in the behavior setting named `DEFAULT_TEXT_REPORT_DELIMITER`

get_user_field_list

This function returns a list of the field and field titles for the security user object in ExtraView.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_user_field_list
```

get_users

This command retrieves the list of users within ExtraView.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_users
&disabled=[Y|N|ONLY]
&filter=pattern
&filter_type=[ID|FIRST|LAST]
```

NOTES

The `disabled` parameter is optional. If provided, the optional values are:

- Y Return disabled users as well as enabled users
- N Return enabled users only. This is the default

ONLY Only return disabled users

The filter parameter is optional. This allows you to perform a wildcard pattern search for specific user records. The wildcard character is an asterisk (*) and you may have more than one of them in the pattern. For example:

*OB Return all records where the user ID ends with the letters OB. For example, this will return BOB but not ROBERT

OB This would return both BOB and ROBERT

The filter_type parameter is optional. If provided, the optional values are as follows. You may provide multiple filter_type parameters in a single get_users call, each with one of the three possible values.

ID This returns the User ID, from the security_user.security_user_id column of the user table

FIRST This returns the first name, from the security_user.first_name column of the user table

LAST This returns the last name, from the security_user.last_name column of the user table

get_valid_meta_data

This action retrieves a complete list of the metadata stored in ExtraView.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=get_valid_meta_data
&user_info=x
&disabled_values=Y
&all=Y
```

NOTES

This function can return a significant amount of data, depending on your installation.

By default, the data returned is for the user's current area and project only.

By default, the data returned is also filtered by any allowed values. For example, if there is an allowed value with the field named STATUS as the parent, then only the child records in the current area will be returned.

The previous two default conditions can be overridden, by using the optional parameter all=Y as part of the call.

The optional parameter `disabled_values` will return all the users, including those that are disabled, within the output to the command.

The optional parameter `user_info` has three possible functions:

1. When no parameter is specified, the command works as documented in the other sections of this page
2. When the parameter has a value of N then no user information is generated in the return of the call
3. When the parameter has a value of Y then one set of generic information is returned, with `SECURITY_USER_ID` being used as the field ID.

The security permissions for each field are checked for the user performing the API command, and only fields to which the user has read permission are returned.

The result of the function is in the form:

`field_namemeta_data_namemeta_data_title`

where the delimiter " is the system delimiter stored in the behavior setting named `DEFAULT_TEXT_REPORT_DELIMITER`.

A small sample of data returned is shown below:

```
CATEGORY|ENHANCEMENT|Enhancement
CATEGORY|HARDWARE|Hardware
CATEGORY|SOFTWARE|Software
OS|5755|FREEBSD
OS|5721|LINUX
OS|5913|NetBSD
OS|5711|SOLARIS
OS|5752|WINDOWS 95
OS|5787|WINDOWS 98
OS|5704|WINDOWS NT
PRIORITY|0|0
PRIORITY|1|1
PRIORITY|2|2
PRIORITY|3|3
PRIORITY|4|4
PRIORITY|5|5
PRIVACY|PRIVATE|Private
PRIVACY|PUBLIC|Public
PRODUCT_NAME|GLOBALINTERACTION|Global Interaction
PRODUCT_NAME|NETTRANSACTIONS|Net Transactions
RELEASE_STATUS|CLOSED|Closed
RELEASE_STATUS|FIXED|Fixed
RELEASE_STATUS|OPEN|Open
RELEASE_STATUS|PENDING|Pending
RELEASE_STATUS|UNASSIGNED|Unassigned
RESOLUTION|CANNOT DUPLICATE|Cannot Duplicate
RESOLUTION|DEFERRED|Deferred
RESOLUTION|DUPLICATE|Duplicate
RESOLUTION|FIXED|Fixed
RESOLUTION|NEED MORE INFO|Need more info
SEVERITY_LEVEL|CRITICAL|Critical
SEVERITY_LEVEL|HIGH|High
SEVERITY_LEVEL|LOW|Low
```

SEVERITY_LEVEL|MEDIUM|Medium

history

The history API command returns all the changes to item records, from a specified point in time to the current time.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=history  
&cutoff=timestamp  
&cutoff_end=timestamp  
&evhist_sellist=selectionList  
&hist_range_end=timestamp  
&hist_range_start=timestamp  
&username_display=ID | LAST | FIRST  
&dd_name_n=value  
&show_attributes=YES | NO
```

NOTES

The timestamp refers to the ITEM.LAST_DATE_UPDATED and the ITEM_HIST.LAST_DATE_UPDATED fields in the database. It is provided within the command in a value of any valid format allowed by ExtraView. If the value supplied can be misconstrued, the user's locale is used to determine the meaning of the date.

The cutoff timestamp provides the current value of the fields. The cutoff timestamp value is not included in the issues generated by this command; that is, the comparison is item timestamp is greater than cutoff timestamp. The cutoff parameter is required.

The cutoff_end parameter is optional. If omitted, there all issues up until the current time are retrieved. If provided, the value must be greater than the value of cutoff. This can be used to limit the items for which history is generated to a specific time period.

The evhist_sellist parameter is a comma-delimited list of field names. This is an optional parameter, and if provided is a list of the fields that are output by the command. If it is not provided, the field list is taken from the detailed report of the user's current business area and project.

hist_range_start and hist_range_end are optional parameters. If omitted, then the items retrieved are generated based upon updates that occurred between these times. If provided then the query becomes a range query, where hist_range_start is the beginning time and hist_range_start is the end time of the updates to the records retrieved.

The optional parameter username_display may be used to override the behavior setting named USERNAME_DISPLAY, for the duration of the execution of a single API call. This allows the developer to return the user names in a different format than the system-wide default.

dd_name_n=value represents an optional list of name value pairs to be used as filters on the query that returns results for the command.

The fields returned by the command are formatted as XML data.

The fields returned in the XML data are subject to two restrictions. The user must have read permission for the field, and the field must exist on the detailed report layout for the user's current area and project settings.

The item fields returned by the command are in the same XML format as those returned by the API get command.

The item fields returned may contain repeating row data within the XML.

If a deleted item record is encountered in the returned data, this will be shown in the XML as follows –

```
<DELETED_RECORD ID='item id' FULL_TIMESTAMP='issue timestamp'/>
```

This command provides a convenient method of determining all changes to the ExtraView database since a point in time, to be used to synchronize data with another ExtraView instance, or with a completely separate application (For example, ExtraView enables the synchronization of data with the Perforce SCM system with this command).

import_allowed_values

This API call imports a tab-delimited file of parent and child values into the ExtraView database. Unlike most API commands (but similar to the `add_attachment` command), this command is designed to be used within an HTML page.

SYNTAX

```
<form method="POST" http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=import_allowed_values
&file=filename
&area=area_id
&project=project_id
&parent=parent_dd_name
&child=child_dd_name
&enctype="multipart/form-data">
</form>
```

The form to be uploaded when prompted by the form must have the following tab-delimited format, where --> represents the tab character:

```
parent_value1 --> child_value1
parent_value1 --> child_value2
parent_value2 --> child_value3
parent_value2 --> child_value4
```

NOTES

`area_id` specifies the `area_id` into which the allowed values will be imported. You can use the Business Area list administration utility in the web interface to see the ID's for all areas.

`project_id` specifies the `project_id` into which the allowed values will be imported. You can use the Project list administration utility in the web interface to see the ID's for all projects.

parent_dd_name specifies the data dictionary name of the parent field which has the allowed value relationship with the specified child.

child_dd_name specifies the data dictionary name of the child field which has the allowed value relationship with the specified parent.

The values in both the parent and child allowed value lists must already exist to work with this command.

EXAMPLE

The following HTML file can be used as a simple template for this command:

```
<html>
  <body>
    <form method="post" action="http://www.mycompany.com/evj/ExtraView/ev_api.action?
      user_id=username&password=password&statevar=import_allowed_values&
      area_id=0&project_id=0&parent=IT_BUILDING&child=IT_BUILDING_FLOOR"
      enctype="multipart/form-data">
      <input type="file" name="file" id="file" size="1" value="" maxlength="256">
      <input type="submit" value="Upload Attachment(s)" class="menuButton"
        title="Click to process and upload the attachments you have selected">
    </form>
  </body>
</html>
```

insert

This API call inserts a new record into the ExtraView database. All fields are treated as optional, and all defined business rules are executed and checked before and after the record is inserted (the preupdate and postupdate rule directives).

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=insert
&p_template_file=file.html
&username_display=ID | LAST | FIRST
&send_email=no
&area=0
&project=0
&short_descr=This%20is%20the%20title
&description=Description%20for%20a%20problem
&status=OPEN
&priority=P1
&assigned_to=jim.smith
&release_found=1.2.3
&product_name=MY_PRODUCT
...
```

There are two name-value pairs that can be provided, that are not fields within ExtraView. These are –

1. &send_email=no

If you supply this name and value, then the insert will override the standard default with which ExtraView will generate email upon the submission of a new issue

2. &username_display=ID | LAST | FIRST

The optional parameter USERNAME_DISPLAY may be used to override the behavior setting named USERNAME_DISPLAY, for the duration of the execution of a single API call. This allows the developer to return the user names in a different format than the system-wide default.

If the issue is added to the database without error, the function will return the issue number as shown below. Note that the term "Bug #" is dependent on the screen title used for the ID field in the data dictionary.

Bug # 12352 added.

NOTES

You must not provide a name and value for the ID field. ExtraView allocates all new issue numbers internally and any attempt to provide an ID will result in an error message, similar to "You cannot provide a Bug # when you are adding a new issue."

Many of the fields within the ExtraView database, such as product_name, status, priority, severity_level, assigned_to, category, etc., must be given valid values that already exist within the meta-data of your installation. If you attempt to enter a value that is not known to ExtraView, an error message will result.

If you attempt to contravene a business rule, an error message will result. For example, if your installation only allows new issues to be SUBMITTED and you immediately attempt to provide a value of CLOSED when inserting a record, an error message will result.

Also, note that special non-alphabetic characters, such as a space, must be "escaped".

You may optionally specify the values for the AREA and PROJECT into which the issue is to be inserted, using their numeric ID. These ID's can be seen within the administration utilities in the web interface. You may not use their titles. If you do not specify the AREA and PROJECT within the parameter list, then the current AREA and PROJECT of the current user are used. For example, use:

```
... ... &p_area=3&p_project=58 ... ..
```

All field permissions are obeyed with the command, and if you attempt to insert a field that does not have write permission, then the whole command will fail.

insert_user

This action creates a new user in the ExtraView database.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=insert_user
&p_template_file=file.html
&security_user_id =userID
&first_name=user_first_name
```

```

&last_name=user_last_name
&security_password=password
&email=email_address
&job_title=user_job_title
&company_name=user_company
&address_line1=user_address_line_1
&address_line2=user_address_line_2
&city=city
&state=state
&postal_code=postal_code
&country=country
&work_telephone=work_telephone
&home_telephone=home_telephone
&cell_phone=cell_phone
&address_line1=user_address_line_1
&fax=fax
&pager=pager

```

NOTES

You must always provide security_user_id, security_password, first_name, last_name and email as fields.

If the ExtraView application default named ENFORCE_DETAILED_USER_INFO has a value of YES, then the COMPANY_NAME, ADDRESS_LINE1, CITY, STATE, POSTAL_CODE and WORK_TELEPHONE must be provided.

The command will fail if the security_user_id already exists.

You must have update permission to the security key named SE_SECURITY_USER before you can execute the command.

insert_xml

This action inserts a new record or records in the ExtraView database from input formatted with XML. The input can be made as part of the HTTP data stream, or can be input from a file in XML format.

SYNTAX

```

http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=insert_xml
&xml_file_name=filename | &xml_string=xml_data
&p_template_file=file.html

```

NOTES

The statevar must be insert_xml.

Provide either the xml_file_name or xml_string, but not both. You provide xml_file_name if the input is from a file that exists at the time of the execution of the command. You provide xml_string, if the data for the insert is provided as part of the HTTP request. This string contains the XML data to be parsed.

p_template_file is the name of the template to be used for return value string generation. Generally, this

template file is stored on the server in the WEB-INF/user_templates directory. On normal completion of the operation, this template undergoes parameter substitution with the following variable names:

Tag	Explanation
<code>__ID__</code>	The item number of the last item inserted
<code>__NUMBER_ITEMS_INSERTED__</code>	The number of inserted items
<code>__ITEM_TITLE__</code>	The title of the ITEM_ID dictionary entry

See the section on Templates for a full explanation of how to create user templates. If no template file is requested, the command returns a completion message to the calling program via HTTP.

On error completion, the return string contains an error message substituted into the error.html user template in the format:

```
error-message "at line=xxx and column number=yyy"
```

where xxx and yyy are the values returned by the XML parser.

As an example, the following message may be returned:

```
"The end-tag for element type "ITEM" must end with a '>' delimiter at line=8 and column number=9"
```

Only one record should be inserted with the XML_STRING in one call to the API. When the input is in a file, there is no restriction to the number of records in a single operation.

The [Administration Guide](#) contains additional information, including the DTD for the XML data and a list of all possible errors.

item_exists

This action checks the ExtraView database for the existence of an issue.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=item_exists
&id=nnnnn
```

NOTES

The output from the call is of the form Defect #nnnnn exists or Defect #nnnnn does not exist.

list_attachment

This action allows you to obtain a list of files attached to an existing record in the ExtraView database. Most importantly, you are able to get a list of the attachment ID's, allowing you to distinguish between the files attached to an issue, for the download action.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=list_attachment
&p_template_file=file.html
&id=nnnnn
```

The return from this command may look like this:

```
12-JUN-2001:index.html:804:Rather, Gary:21:The index.html as revised
14-JUN-2001:Applic.doc:28160:Koppel, Carl:38:Application notes
```

NOTES

The delimiter is the value of the DEFAULT_TEXT_REPORT_DELIMITER in the ExtraView application defaults.

The order of the fields returned is date of upload, filename, file size (in bytes), name of person who uploaded the attachment, ID of the attachment and the description.

run_report

This function runs an existing report, using its report_id obtained from the get_reports function.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=run_report
&username_display=ID | LAST | FIRST
&api_reverse_lookup=NO | YES
&id=report_id
&page_length=100
&record_start=1
&record_count=120
&persist_handle=xxx
&field1=value1
&field2=value2
```

NOTES

This command runs a report from the available list of public and private reports using the report_id obtained from the get_reports function.

The output is returned in XML format.

The fields returned in the results correspond to the fields on the detailed report layout of the user's

current business area, current project and current role. However, if the behavior setting named `REPORT_DTL_ITEM_DATA_LAYOUT` is set to YES, then the detailed report for the user's current role and the issue's business area and project are used to define the fields being returned.

The parameter named `page_length` is required and gives the ExtraView API the maximum number of records to return with one call. In conjunction with the parameter named `p_record_start`, you can build JavaScript functions to retrieve paginated results, if you believe your searches can bring up large number of records.

The parameter named `record_start` is required and gives ExtraView the number of the first record in the search results to display. This is used in conjunction with the parameter named `p_page_length`, which defines the number of results to retrieve. If there is a possibility that `record_start` can be greater than `page_length`, you must use the parameter named `record_count`. Using these parameters, you can build functions to retrieve paginated results if you believe your searches can bring up large number of records.

The optional parameter `api_reverse_lookup` has a default value of NO. If set to YES, then all the user defined field values are expressed with the field's title as opposed to the field's ID. This is useful when using fields with a display type of list, where the programmer needs to know the internal ID of a list value in order to use it as a parameter in the API call. With `api_reverse_lookup=YES`, then the programmer can use the field's title. Usually this is well known and obvious as opposed to ID's which need to be discovered through a SQL query on the database.

The optional parameter `username_display` may be used to override the behavior setting named `USERNAME_DISPLAY`, for the duration of the execution of a single API call. This allows the developer to return the user names in a different format than the system-wide default.

The optional parameter named `persist_handle` may be used to paginate the result set by spreading the results returned over several separate calls using this API command. The `persist_handle` identifies the result set, qualified by the `user_id`, that is used to maintain the result set information across multiple API calls. The rules for using `persist_handle` are as follows:

- A new result set is generated whenever `record_start=1`; an existing result set is used whenever `record_start > 1`
- The same `persist_handle` value can only be used for subsequent pages of the result set on the same node; in clustered environments, if a subsequent call is directed to another node, the result set will not be found and results will not be returned
- The maximum number of pages returned is 10. This means that the `page_length` must be greater than 1/10th the size of the result set (# of rows returned) or the results will be truncated with no error indication.

The optional parameters represented by `field1=value1` and `field2=value2` are used to provide any number of runtime filters that are required by the report. For a name value pair to be valid with this API call, the field must have been defined within the report as a runtime filter. You may omit runtime filters that were defined in the report if you do not require them. This has the effect of not using that field as a filter at all.

search

This API call allows you to search the ExtraView database and to return a set of records that match the search criteria. This function is equivalent to the search capability within the browser version of ExtraView. It is extremely powerful as multiple search filters can be set on different fields. For example, it is straightforward to set up a search that responds to a query such as "tell me all the open issues against a specific module within a specific product that contain a specific keyword.

SYNTAX

```

http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=search
&page_length=100
&record_start=1
&record_count=10
&p_template_file=file.html
&persist_handle=xxx
&username_display=ID | LAST | FIRST
&status=OPEN
&module_id=WIDGET
&product_name=MY_PRODUCT
&keyword=wireless%20PDA
. . .

```

For example, a return from a valid search may be as shown in the following XML:

```

<?xml version="1.0"?>
<EXTRAVIEW_RESULTS>
<PROBLEM_RECORD>
<ID TITLE="Bug #">12266</ID>
<SHORT_DESCR TITLE="Title"><![CDATA[Here is the title]]></SHORT_DESCR>
<SEVERITY_LEVEL TITLE="Severity">02</SEVERITY_LEVEL>
<PRIORITY TITLE="Priority">2</PRIORITY>
<PRODUCT_NAME TITLE="Product">NetOp</PRODUCT_NAME>
<DATE_CREATED TITLE="Created">19-APR-01</DATE_CREATED>
<OWNER TITLE="Owner">DIAMONDK</OWNER>
<TIMESTAMP TITLE="Last Modified">21-JUN-01</TIMESTAMP>
<ASSIGNED_TO TITLE="Assigned To">CARL.KOPPEL</ASSIGNED_TO>
<PRIVACY TITLE="View">Private</PRIVACY>
<CATEGORY TITLE="Category">Software</CATEGORY>
<RESOLUTION TITLE="Disposition"></RESOLUTION>
<DATE_CLOSED TITLE="Date Closed"></DATE_CLOSED>
<ORIGINATOR TITLE="Originator">ROBBIE.LLOYD</ORIGINATOR>
<ORIGINATOR_NAME TITLE="Originator">Rob Lloyd</ORIGINATOR_NAME>
<MODULE_RECORD>
<MODULE_NAME TITLE="Module">Server</MODULE_NAME>
</MODULE_RECORD>
<RELEASE_RECORD>
<RELEASE_FOUND TITLE="Version Open">1.0.1.16</RELEASE_FOUND>
<RELEASE_FIXED TITLE="Version Closed"></RELEASE_FIXED>
<RELEASE_STATUS TITLE="Release Status">Open</RELEASE_STATUS>
</RELEASE_RECORD>
</PROBLEM_RECORD>
<PROBLEM_RECORD>
<ID TITLE="Bug #">12118</ID>
<SHORT_DESCR TITLE="Title"><![CDATA[Another problem]]></SHORT_DESCR>
<SEVERITY_LEVEL TITLE="Severity">01 sev</SEVERITY_LEVEL>
<PRIORITY TITLE="Priority">1</PRIORITY>
<PRODUCT_NAME TITLE="Product">NetOp</PRODUCT_NAME>
<DATE_CREATED TITLE="Created">14-FEB-01</DATE_CREATED>
<OWNER TITLE="Owner">CARL.KOPPEL</OWNER>
<TIMESTAMP TITLE="Last Modified">21-APR-01</TIMESTAMP>
<ASSIGNED_TO TITLE="Assigned To">CARL.KOPPEL</ASSIGNED_TO>
<PRIVACY TITLE="View">Private</PRIVACY>

```



```

<CATEGORY TITLE="Category">Software</CATEGORY>
<RESOLUTION TITLE="Disposition"></RESOLUTION>
<DATE_CLOSED TITLE="Date Closed"></DATE_CLOSED>
<ORIGINATOR TITLE="Originator">CARL.KOPPEL</ORIGINATOR>
<ORIGINATOR_NAME TITLE="Originator">Carl Koppel</ORIGINATOR_NAME>
<MODULE_RECORD>
<MODULE_ID TITLE="Module">Client</MODULE_NAME>
</MODULE_RECORD>
<RELEASE_RECORD>
<RELEASE_FOUND TITLE="Version Open">Framework 1.1</RELEASE_FOUND>
<RELEASE_FIXED TITLE="Version Closed"></RELEASE_FIXED>
<RELEASE_STATUS TITLE="Release Status">Closed</RELEASE_STATUS>
</RELEASE_RECORD>
</PROBLEM_RECORD>
</EXTRAVIEW_RESULTS>

```

Note that if you do not have permission to view any of these fields, they will not appear in the output from the action.

This action purposely returns only a small number of fields from the database. If you require additional fields, you can parse the ID out of the returned information and then use the get action to read the remaining fields within the database.

You should be careful in your use of this action as it can conceivably return extremely large result sets to you.

NOTES

- The fields returned in the results correspond to the fields on the detailed report layout of the user's current business area, current project and current role. However, if the behavior setting named REPORT_DTL_ITEM_DATA_LAYOUT is set to YES, then the detailed report for the user's current role and the issue's business area and project are used to define the fields being returned
- The keywords parameter is not a database field, but can be used to provide an unlimited number of keywords as search filters in a space-delimited list to ExtraView. Note that you must "escape" characters such as spaces in this list
- The parameter named p_page_length is required and gives the ExtraView API the maximum number of records to return with one call. In conjunction with the parameter named p_record_start, you can build script functions to retrieve paginated results, if you believe your searches can bring up large number of records
- The parameter named p_record_start is required and gives ExtraView the number of the first record in the search results to display. In conjunction with the parameter named p_page_length, you can build script functions to retrieve paginated results, if you believe your searches can bring up large number of records
- You need to supply the parameter named p_record_count whenever p_record_start is greater than p_page_length
- The parameter named p_template_file is optional. If it is not provided, ExtraView returns the results of the query in XML format. If it is provided, its value is the name of a server-side file that contains a template to format the results of the query. Using this template provides a means of formatting the output from the search command to a style of your own choosing. Most commonly, this is used to provide a style of output consistent with that of the remainder of your own company's web site. An example of the source to a template file is shown below.

```

<TABLE cellpadding="2" cellspacing="2" border="1" bordercolor="#FFCCCC">
<TR bgcolor="#CCCCFF">
<TD align=right width=80><font size="-1">Defect # </font></TD>
<TD width=800><font size="-1">__TAG_ID__</font></TD>

```

```

</TR>
<TR>
<TD align=right><font size="-1">Title</font></TD>
<TD><font size="-1">__TAG_SHORT_DESCR__</font></TD>
</TR>
<TR>
<TD align=right><font size="-1">Product</font></TD>
<TD><font size="-1">__TAG_PRODUCT_NAME__</font></TD>
</TR>
<TR>
<TD align=right><font size="-1">Description</font></TD>
<TD>
<!-- __DESCRIPTION__ -->
<PRE>
__TAG_DESCRIPTION_TEXT__
</PRE>
</TD>
</TR>
<TR>
<TD align=right><font size="-1">Comments</font></TD>
<TD>
<!-- __COMMENTS__ -->
<PRE>
<b>__TAG_COMMENTS_USER__: __TAG_COMMENTS_TIMESTAMP__</b>
<br>
__TAG_COMMENTS_TEXT__
</PRE>
</TD>
</TR>
</TABLE>

```

- The optional parameter `username_display` may be used to override the behavior setting named `USERNAME_DISPLAY`, for the duration of the execution of a single API call. This allows the developer to return the user names in a different format than the system-wide default
- If no records are returned by the search, the message "No records found." will be displayed
- You must provide at least one name and value for the search criteria. If you do not do this, you will see the message "No parameters entered"
- Many of the fields within the ExtraView database, such as `product_name`, `status`, `priority`, `severity_level`, `assigned_to`, `category`, etc., must be given valid values that already exist within the metadata of your installation. If you attempt to enter a value that is not known to ExtraView, an error message will result
- The optional parameter named `persist_handle` may be used to paginate the result set by spreading the results returned over several separate calls using this API command. The `persist_handle` identifies the result set, qualified by the `user_id`, that is used to maintain the result set information across multiple API calls. The rules for using `persist_handle` are as follows:
 - A new result set is generated whenever `record_start=1`; an existing result set is used whenever `record_start > 1`
 - The same `persist_handle` value can only be used for subsequent pages of the result set on the same node; in clustered environments, if a subsequent call is directed to another node, the result set will not be found and results will not be returned
 - The maximum number of pages returned is 10. This means that the `page_length` must be greater than 1/10th the size of the result set (# of rows returned) or the results will be truncated with no error indication.
- The format of date filter parameters, as used in runtime filters in a report is as follows:

```
<date> || <date> - <date> || -<date> || <date>-
```

The latter three are date ranges; rangestart to rangestop, rangestop, and rangestart respectively.

Where <date> is:

<unquoted date> || <sq><unquoted date><sq> || <dq><unquoted date><dq>

where <dq> ::= " (a double quote)

and <sq> := ' (a single quote)

A date may contain a dash if it appears in quotes. Otherwise, a dash is not permitted except as a date range signifier.

search_field_list

This command provides a list of fields that may be used as query filters for the current user.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=search_field_list
```

NOTES

The delimiter used the results is the value of the DEFAULT_TEXT_REPORT_DELIMITER in the ExtraView application defaults.

This command takes into account all security permissions for the user.

The general form of each entry returned by the command is:

<prefix><fieldName><delimiter><fieldTitle><delimiter><parentName>

- The parentName is blank or the immediate parent in an allowed-value or database-type relationship
- The delimiter is that specified in in the behavior setting named DEFAULT_TEXT_REPORT_DELIMITER
- The characters in the prefix have the following meanings:
 - ~ means the field is part of a repeating row layout
 - * means the field has a display type of textarea or logarea
 - % means the field has a display type of user

An example of the return from this command is:

```
TIMESTAMPLast Modified
EMAIL_SWITCHGenerate Email
%CHANGED_BYChanged By
KEYWORDKeywords
PRODUCT_NAMEProduct
*SUGGESTIONSEngineering Remarks
RELEASE_FOUNDRelease
SHORT_DESCRTitle
SEVERITY_LEVELSeverity
```

PRIORITYPriority
STATUSStatus
IDDefect #
DATE_CREATEDCreated
%OWNEROwner
CATEGORYCategory
RESOLUTIONResolution
%LAST_CHANGE_USERChanged by
DAYS_IN_STATUSDays in Status
DAYS_OPENDays Open
MONTHS_OPENMonths Open
%ASSIGNED_TOAssigned To
%ORIGINATOROriginator
PRIVACYView
RELEASE_FIXEDRelease Fixed
MONTHS_IN_STATUSDays In Status
START_UPDATEUpdated Start Date
WEEKS_IN_STATUSDays In Status
WEEKS_OPENWeeks Open
BUILD_FOUND_INBuild Found In
BUILD_FIXED_INBuild Fixed In

set_area_proj

This command sets the working area and project for the current user.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=set_area_proj  
&area_id=nnn1  
&proj_id=nnn2
```

NOTES

Both the area_id and the project_id must exist, else the command will fail.

The proj_id must be valid within the area_id, else the command will fail.

set_role

This action sets the user role of the current user.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=set_role
```

&user_group=role_id

NOTES

The role_id specified, must exist, else the command will fail.

You can get a list of the valid roles with the command get_roles.

update

This command allows you to update an existing record within ExtraView's database. Only values that are being altered need be supplied within the API call. Fields whose values are to remain the same need not be supplied as parameters. All defined business rules using the preupdate and the postupdate directives are executed and checked before and after the record is updated.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=update
&p_template_file=file.html
&id=nnnnn
&status=FIXED
&release_fixed=1.2.3
. . .
```

There is one additional name and value pair that can be provided, that is not a field within ExtraView. This is

&send_email=no. If you supply this name and value pair, then the update will override the set default with which ExtraView will generate email upon the update of an existing issue.

If the issue is updated without error, the function will return the issue number as shown below. Note that the term "Bug #" is dependent on the screen title used for the ID field in the data dictionary.

Bug # 12352 updated.

NOTES

You must provide a valid name and value pair for the ID field. The value must be an existing issue ID within the ExtraView database. If you do not provide this, an error message is generated.

Many of the fields within the ExtraView database, such as product_name, status, priority, severity_level, assigned_to, category, etc., must be given valid values that already exist within the metadata of your installation. If you attempt to enter a value that is not known to ExtraView, an error message will result.

If you attempt to contravene a business rule, an error message will result. For example, if your installation only allows issues to be FIXED if they are in an OPEN state and you provide a value of CLOSED, an error message will result.

All field permissions are obeyed with the command, and if you attempt to update a field that does not have write permission, then the whole command will fail.

The Role of PROBLEM_RELEASE_ID in the update Command

Usually, when one or more specific repeating rows must be affected (modified or deleted) through the update command, the `PROBLEM_RELEASE_ID` parameter is used to identify specific rows. More than one instance of `PROBLEM_RELEASE_ID` may be specified. In this case, each *field=value* parameter corresponds to the `PROBLEM_RELEASE_ID` which shares the same position in the parameter list for the same field name.

Thus, for example,

```
...&problem_release_id=111&problem_release_id=222&rr_field=abc&rr_field=def
```

maps the `rr_field=abc` to `problem_release_id 111` and `rr_field=def` to `problem_release_id 222`.

When no `PROBLEM_RELEASE_ID` in the parameter list maps to a specific `rr_field=value`, then a new repeating row is added to accommodate the new value.

Multiple Repeating Row Types

ExtraView supports repeating rows of multiple types. Each type denotes a grouping of fields based on a layout with that *item group type*. All repeating row types can be updated via the API.

There are two ways of specifying a repeating row type:

- Specific `PROBLEM_RELEASE_IDxxx` values, where `xxx` is the item group type, e.g.,
`PROBLEM_RELEASE_ID3`
- Generic `PROBLEM_RELEASE_ID` values.

Note1: Specific and generic `PROBLEM_RELEASE_ID`'s may not be combined in a single API call.

Note2: Each API Update operation using generic `PROBLEM_RELEASE_ID` values can support only one repeating row type. Attempts to mix two or more repeating row types in a single API operation with generic `PROBLEM_RELEASE_ID`'s will return with an exception.

The *edit* screen layout, along with the repeating row layouts applicable to the update request, define which fields belong to which item group type, and therefore, which `PROBLEM_RELEASE_IDxxx`'s apply.

When generic `PROBLEM_RELEASE_ID`'s are used, all the specified fields must be grouped into the same item group type (via the edit layout and the repeating row layouts) – this item group type is used to qualify the `PROBLEM_RELEASE_ID` values.

Example of specific `PROBLEM_RELEASE_ID` Update

```
...&PROBLEM_RELEASE_ID3=111&PROBLEM_RELEASE_ID4=222&field1=val1&field2=val2
```

where `field1` is in the repeating row layout of item group type 3 and `field2` is in the repeating row layout of item group type 4, both of which are embedded in the edit layout for the user's current business area, project and role. Then two repeating rows are updated, one with `field1=val1` and the other with `field2=val2`, each in a different repeating row type.

Example of generic `PROBLEM_RELEASE_ID` Update

```
...&PROBLEM_RELEASE_ID=111&field1=val1
```

where `field1` is in the repeating row layout of item group type 3 and repeating row 111 is in item group type 3 results in the update of `field1` in the specified row. Note that there is no item group type specified in the `PROBLEM_RELEASE_ID` variable name.

Using update to Add Repeating Rows

To add new repeating rows, values for fields in the repeating rows are provided without a corresponding PROBLEM_RELEASE_ID value. For each non-corresponding value of a field, a new repeating row is inserted. Thus, specifying &rr_field_a=1&rr_field_a=2&rr_field_a=3 with no specification of PROBLEM_RELEASE_ID will result in three new repeating rows being added with these values, assuming rr_field_a is a field in the repeating row. To update a repeating row value, you must provide the PROBLEM_RELEASE_ID value of the row in question.

Using update to Delete Repeating Rows

Within the ExtraView GUI, users check a box named PROBLEM_RELEASE_DELETE in order to delete a repeating row. The API can emulate this behavior, by setting PROBLEM_RELEASE_DELETE to the checked for value for any repeating row you wish to delete.

For example, use a call similar to the following to delete a repeating row:

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=update
&id=nnnnn
&problem_release_id=xxxxx
&problem_release_delete=checked
```

update_user_password

This action allows the user to update the password of an existing user within ExtraView.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=update_user_password
&p_template_file=file.html
&security_user_id=user_name
&old_password=old_password
&new_password=new_password
```

NOTES

You must provide the ID of an existing user in the security_user_id parameter.

You must provide both the old password and the new password. These will be checked internally within ExtraView to ensure they conform to any rule that may be in place.

You must have update permission to the security key named SE_SECURITY_USER before you can execute the command.

user_field_list

This action allows the user to retrieve all the fields that are part of the user record. It is typically used to

generate a list of fields for which to provide values when creating a new user.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=user_field_list
&security_user_id=user_name
```

NOTES

You must have query permission to the security key named SE_SECURITY_USER before you can execute the command.

The list returned will show a plus character (+) if the field is required.

A typical output from the command is shown below.

```
+SECURITY_USER_ID|Security User Id
+SECURITY_PASSWORD|Security Password
+FIRST_NAME|First Name
+LAST_NAME|Last Name
JOB_TITLE|Job Title
COMPANY_NAME|Company
ADDRESS_LINE1|Address Line1
ADDRESS_LINE2|Address Line2
CITY|City
STATE|State
POSTAL_CODE|Postal Code
COUNTRY|Country
+EMAIL|E-Mail Address
WORK_TELEPHONE|Work Telephone
HOME_TELEPHONE|Home Telephone
CELL_PHONE|Cell Phone
FAX|Fax
PAGER|Pager
```

user_group_list

This action allows the user to retrieve a list of all the user groups within the system. It is typically used to generate a list that validates adding a user to a user group.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?
user_id=username
&password=password
&statevar=user_group_list
&security_user_id=user_name
```

NOTES

You must have query permission to the security key named SE_SECURITY_GROUP before you can execute the command.

A typical output from the command is shown below.

```
ADMIN:Administrator  
CUSTOMER:Customer  
HW_ENG:HW Engineering  
QA:Quality Assurance  
SW_ENG:SW Engineering  
SUPPORT:Support
```

version

This action returns the build information of your ExtraView database.

SYNTAX

```
http://www.myserver.com/evj/ExtraView/ev_api.action?  
user_id=username  
&password=password  
&statevar=version
```

NOTES

The information returned is a serial number of the last updates applied to the ExtraView database.

A typical output from the command is shown below.

```
$Revision: 22 $ $Modtime: 6/23/06 10:37p $
```