REST APIs for Personal Data/Control

Process Dashboard 2.4.1 includes a large number of new REST APIs. These APIs make it possible for an external process to query and modify personal data, as well as GUI state. This document describes those new APIs.

These APIs were originally developed to support the needs of a Visual Studio (VS) plugin, so the explanations in the documentation below are often written with that target in mind. But of course, a variety of external processes can access these APIs. For example, the APIs could be used to build a plugin for a different IDE, or to enable remote control from other desktop tools.

Scope

The REST APIs in this document are published by the personal Process Dashboard toolbar. They are only available while the personal dashboard is running, and only provide access to the personal data in a single person's database. As such, their target is to enable integration with other tools to simplify an individual's personal workflow.

To access team data and rollups, a better entry point would be the Team Process Data Warehouse. That warehouse is published by the Team Dashboard, and can also be instantiated on demand from a command line. See the Team_Process_Data_Warehouse documentation for more information.

Security

These REST APIs are published by the Process Dashboard's internal web server, which only listens on a "localhost" URL by default. Thus, a process would need to be running on your local computer to be able to connect to these REST APIs. Other development tools like IDEs or command-line batch scripts generally fit this model well.

This restriction prevents an individual elsewhere on your network from connecting to these APIs to read or modify your data. But it also prevents you from accessing these REST APIs from a build server or other remote system. Any architecture which interacts with these REST APIs will need to have a component running on the same computer as the Process Dashboard.

CORS is not currently enabled for these REST APIs; so interactions with them from a web browser will be subject to the same-origin policy. If any developers are interested in using these REST APIs from a web mashup, please contact the Process Dashboard development team, so we can work together to design the appropriate security controls.

Installation

To obtain access to these REST APIs, download Process Dashboard 2.4.1 from:

http://www.processdash.com/download#beta

Timer API

http://localhost:2468/api/v1/timer/

GET

A GET request to this URL will return a JSON document with information about the current state of the dashboard timer. This includes whether the timer is running or paused, and which task is currently selected. Here is an example of the output:


```
{
  "timer": {
      "timing": false,
      "timingAllowed": true,
      "defectsAllowed": true,
      "activeTask": {
          "id": "7",
          "fullName": "Component A/Subcomponent B/Code Review",
          "project": {
              "id": "iw9w2hlv",
              "name": "Team Project",
              "fullName": "/Project/Team Project",
              "creationDate": "2016-12-03T17:07:27.907-0700"
          },
          "completionDate": "2017-08-06T22:45:18.411-0600",
          "estimatedTime": 120.4,
          "actualTime": 182
      }
  },
  "stat": "ok"
}
```


Notes:

- The top-level JSON document includes two elements: a “stat” key indicating “ok” for success, and a “timer” key with the actual data.
- The “timer” object includes a “timing” property which is true if the timer is running, false if paused.
- The “timer” object includes a “timingAllowed” property which is usually true, but could be false if timing is forbidden for the active task.
- The “timer” object includes a “defectsAllowed” property which is usually true, but could be false if defect logging is forbidden for the active task.
- The “timer” object includes an “activeTask” property identifying the task that is currently active, which time will be logged to. See the Task API below for more information about the values in this object.

PUT

The VS plugin can make a PUT request to the Timer API URL. The body of the PUT should be a set of key/value pairs in application/x-www-form-urlencoded format. Accepted parameters keys in the PUT body are:

Parameter Key	**Value**
`timing`	`true` to start the timer, `false` to pause. (Optional; if omitted, the state of the timer will not be changed.)
`activeTaskId`	The identifier of a different task that should become active. (Optional; if omitted, the active task will not be changed.)

Task List API

http://localhost:2468/api/v1/tasks/

GET

The VS plugin can issue a GET request to retrieve a list of all “leaf” tasks known to this dashboard. (“Leaf” tasks are tasks that have no children, where time can be logged.) Here is an example of the output:


```
{
  "tasks": [
    {
      "id": "3",
      "fullName": "Software Component/Strategy",
      "project": {
        "id": "iw9w2hlv",
        "name": "Team Project",
        "fullName": "/Project/Team Project",
        "creationDate": "2016-12-03T17:07:27.907-0700"
      },
      "completionDate": "2017-03-15T19:32:04.000-0700"
    },
    {
      "id": "4",
      "fullName": "Software Component/Code",
      "project": {
        "id": "iw9w2hlv",
        "name": "Team Project",
        "fullName": "/Project/Team Project",
        "creationDate": "2016-12-03T17:07:27.907-0700"
      },
      "completionDate": "2017-04-15T19:32:04.000-0700"
    },
    {
      "id": "7",
      "fullName": "Software Component/Test",
      "project": {
        "id": "iw9w2hlv",
        "name": "Team Project",
        "fullName": "/Project/Team Project",
        "creationDate": "2016-12-03T17:07:27.907-0700"
      }
    },
    {
      "id": "1",
      "fullName": "Non Project",
      "project": {
        "id": "0",
        "name": "Other Tasks",
        "fullName": "",
        "creationDate": "1969-12-31T17:00:00.000-0700"
      }
    }
  ],
  "stat": "ok"
}
```


The document includes a “stat” key indicating success and a “tasks” key with an array of task objects. Each task object contains an id, a fullName, and a project object. If the task has been marked complete, its completionDate key will also be set. Other details about the task (its planned and actual time, etc) are omitted to keep the resulting document from growing too large.

Recent Tasks API

http://localhost:2468/api/v1/tasks/recent/

GET

A GET request to this URL will return a JSON document with information about tasks where time has been logged recently. Here is an example of the output:


```
{
  "recentTasks": [
    {
      "id": "70",
      "fullName": "Component A/Test",
      "project": {
        "id": "hllesl3q",
        "name": "Team Project",
        "fullName": "/Project/Team Project",
        "creationDate": "2013-09-14T15:30:56.726-0700"
      },
      "completionDate": "2014-08-09T12:12:37.318-0700",
      "estimatedTime": 6000,
      "actualTime": 5940
    },
    {
      "id": "73",
      "fullName": "Component A/Code",
      "project": {
        "id": "hllesl3q",
        "name": "Team Project",
        "fullName": "/Project/Team Project",
        "creationDate": "2013-09-14T15:30:56.726-0700"
      },
      "completionDate": "2014-08-09T10:58:36.724-0700",
      "estimatedTime": 6000,
      "actualTime": 9300
    },
    {
      "id": "74",
      "fullName": "Component A/Design",
      "project": {
        "id": "hllesl3q",
        "name": "Team Project",
        "fullName": "/Project/Team Project",
        "creationDate": "2013-09-14T15:30:56.726-0700"
      },
      "completionDate": "2014-08-09T12:12:34.470-0700",
      "estimatedTime": 6000,
      "actualTime": 3300
    }
  ],
  "stat": "ok"
}
```


The document includes a “stat” key indicating success and a “recentTasks” key with an array of task objects. For a description of these objects, see the Task Details API below. The first item in the array is the task where time was logged most recently, followed by the second most recent, and so on. The tasks listed could potentially come from multiple projects.

This API can accept an optional query parameter, as shown below:

Parameter Key	**Value**
`maxResults`	The maximum number of tasks to return. (Optional: if omitted, the default value of 10 is used.) The response could contain fewer than the requested number of items, if this individual hasn’t worked on that many tasks.

Task Details API

http://localhost:2468/api/v1/tasks/{taskId}/

GET

The VS plugin can issue a GET request to retrieve the details about a particular task. The {taskId} placeholder in the URL above should be replaced with the ID of the task in question. Here is an example of the output:


```
{
  "task": {
      "id": "7",
      "fullName": "Component A/Subcomponent B/Code Review",
      "project": {
          "id": "iw9w2hlv",
          "name": "Team Project",
          "fullName": "/Project/Team Project",
          "creationDate": "2016-12-03T17:07:27.907-0700"
      },
      "completionDate": "2017-08-06T22:45:18.411-0600",
      "estimatedTime": 120.4,
      "actualTime": 182
  },
  "stat": "ok"
}
```


The document includes a “stat” key indicating success and a “task” key with an object of data:

- The “id” key provides the ID of this task. This should be treated as a string; there is no guarantee it will always look like a number.
- The “fullName” key provides the complete, hierarchical name of this task within the enclosing project.
- The “project” key provides details about the project that contains this task. For more information about the data in this object, see the Project Details API below.
- The “completionDate” key provides the date this task was completed, in ISO 8601 format. If this task has not be completed, this key will not be present.
- The “estimatedTime” key provides the estimated time for this task, in minutes. This could be a floating point number.
- The “actualTime” key provides the actual time logged against this task, in minutes.

If the request contains a query parameter of the form “expand=resources”, the task object will also include a “resources” key giving the data that would be returned by the Task Resources API.

If the request contains a query parameter of the form “expand=children”, the task object will also include a “children” key, listing the immediate children of this task. (The “children” list will be empty if this is a leaf task.) For brevity, child tasks will only contain:

- A “name” field (instead of “fullName”), indicating their name relative to their parent
- An “id” field, which can be used in subsequent calls to this (or other) APIs

If the query parameter is of the form “expand=descendants”, the “children” key will be present, and the items in that list will also contain children, recursively. Special notes:

- The root node of a project has an ID of “projectID:root”, so a hierarchy of the tasks in a project can be retrieved with /api/v1/tasks/{projectID}%3Aroot/?expand=descendants
- The root of the entire hierarchy has the ID “0:root”, so the entire task tree can be retrieved with /api/v1/tasks/0%3Aroot/?expand=descendants

PUT

The VS plugin can make a PUT request to the URL for a particular task. The body of the PUT should be a set of key/value pairs in application/x-www-form-urlencoded format. Accepted parameters keys in the PUT body are:

Parameter Key	**Value**
`completionDate`	`null` or the empty string to mark the task incomplete `now` to mark the task complete with the current date/time A date in ISO 8601 format to mark the task complete with a given date (Optional: if omitted, the completion date will not be changed.)
`estimatedTime`	A new estimated time for this task, in minutes. (Optional: if omitted, the estimated time will not be changed)

Task Resources API

http://localhost:2468/api/v1/tasks/{taskId}/resources/

GET

The Process Dashboard provides a large number of reports, scripts, forms, and other tools that provide project- and task-specific functionality via URLs. The VS plugin can retrieve a list of the resources for a given task by issuing a GET request to this API. The {taskId} placeholder in the URL above should be replaced with the ID of the task in question. Here is an example of the output:

```
{
  "resources": [
    {
      "name": "TSP(SM) Project Plan Summary",
      "uri": "/Project/Team+Project//cms/TSP/indiv_plan_summary",
      "taskPath": "/Project/Team Project"
    },
    {
      "name": "Workflow Process Analysis",
      "uri": "/Project/Team+Project//reports/workflowToDate?wait",
      "taskPath": "/Project/Team Project"
    },
    {
      "name": "Project Task & Schedule",
      "uri": "/Project/Team+Project//control/showTaskSchedule.class?trigger&TSP-indiv",
      "taskPath": "/Project/Team Project",
      "trigger": true
    },
    {
      "name": "Project Parameters and Settings",
      "uri": "/Project/Team+Project//TSP/indiv_project_parameters.shtm",
      "taskPath": "/Project/Team Project"
    }
  ],
  "stat": "ok"
}
```

The resulting items would be appropriate to display in a pop-up menu in the order listed, using the “name” field as the menu item text to display to the user. When the user chooses one of these items from the menu:

- If the chosen item does not have a “trigger” key, an absolute URL should be constructed by adding “http://localhost:2468” to the beginning of the value in the “uri” field. Then, a new web browser tab should be opened in Visual Studio to display the given URL.
- If the chosen item has a “trigger” key with a value of true, the VS plugin should retrieve the value from the “uri” field and use it in a call to the Trigger API. The Trigger API will respond with instructions about the appropriate action to take.

Notifications API

http://localhost:2468/api/v1/notifications/

GET

Sometimes, an event will occur in Process Dashboard that requires the user's attention. (For example, after another individual alters the team plan, a person may be alerted of the need to perform a "Sync to WBS" operation.)

If a user keeps the Process Dashboard hidden and only interacts with their IDE, the dashboard will never have an opportunity to display alerts about these events. Accordingly, the IDE will need to display the alerts on behalf of the dashboard.

At any time, VS can make a call to this API to retrieve a list of the user notifications that are waiting. Here is an example of the output:

```
{
  "notifications": [
    {
      "id": "teamdash.SyncScanner:/Project/Team Project",
      "message": "The project 'Foo' needs to be synchronized to the team WBS.",
      "uri": "/Project/Team+Project//TSP/setup/sync.class?"
    },
    {
      "id": "joinTeamProject.j26sgvcz",
      "message": "You have been invited to join the 'Foo I2' project.  Double-click
                 here to either accept or decline the invitation.",
      "uri": "/Team_Project_Invite/0//dash/teamStart.class?page=invite"
    }
  ],
  "stat": "ok"
}
```

When notifications are present, VS might wish to display an indicator with the number of notifications that are waiting. If the user clicks on that indicator, the notification messages can be displayed in a list.

When the user clicks on one of those notifications, VS should open the uri for the given notification. (This uri could be a regular URI or a trigger, and should be opened as described in the previous section.) After the notification has been handled, it will no longer appear in the list returned by this API.

The VS IDE can use the Event Listener API to discover when new notifications arrive. When that API delivers an event of type "notification", it is appropriate to call this API and get the current notification list.

Trigger API

http://localhost:2468/control/runTrigger

POST

As mentioned in the previous section, the Process Dashboard provides a number of URL-based tools that provide project- and task-specific functionality. Most of these resources have an HTML-based user interface which can be readily displayed in a web browser control. But some tools can be better served with a rich-client user interface. Those resources are labeled as "trigger" resources by the API above.

When the user chooses to open a trigger resource, the VS plugin should make a POST request to this API. The URL-encoded POST body should include a key named "uri" with the value of the trigger resource the user wishes to run. This API will respond with a JSON document which will take **one** of the forms below.

Window Response

```
{
  "window": {
       "id": 183066,
       "pid": 1234,
       "title": "Task & Schedule"
  },
  "stat": "ok"
}
```

The Process Dashboard has opened a window in response to this trigger. The VS plugin should bring the new window to the front of the VS IDE, as described below.

Message Response

```
{
  "message": {
       "title": "Synchronization Complete",
       "body": "Your personal plan is up to date."
  },
  "stat": "ok"
}
```

The user should be shown a dialog with a specific title and message body. The dialog should have an "OK" button for dismissing the message.

URL Redirect Response

```
{
  "redirect": "http://localhost:2468/Project/Team+Project//TSP/setup/sync.class",
  "stat": "ok"
}
```

After considering the trigger, the dashboard has determined that an HTML-based user interface is appropriate after all. The VS plugin should open a new web browser control to display the given URL.

Project List API

http://localhost:2468/api/v1/projects/

GET

The VS plugin can issue a GET request to retrieve high level information about all of the projects in this personal dashboard. Here is an example of the output:


```
{
  "projects": [
    {
      "id": "ixnnzi9l",
      "name": "Team Project 2",
      "fullName": "/Project/Team Project 2",
      "creationDate": "2017-01-07T13:09:40.665-0700"
    },
    {
      "id": "iw9w2hlv",
      "name": "Team Project",
      "fullName": "/Project/Team Project",
      "creationDate": "2016-12-03T17:07:27.907-0700"
    },
    {
      "id": "iokdum2d",
      "name": "Mobile App I1",
      "fullName": "/Project/Mobile/Mobile App I1",
      "creationDate": "2016-05-23T13:05:16.597-0600"
    },
    {
      "id": "0",
      "name": "Other Tasks",
      "fullName": "",
      "creationDate": "1969-12-31T17:00:00.000-0700"
    }
  ],
  "stat": "ok"
}
```


The document includes a “stat” key indicating success and a “projects” key with an array of data. Each element in the array represents a separate project in this personal dashboard. This array will always include at least one entry. For details about these objects, see the Project Details API section below.

Project Details API

http://localhost:2468/api/v1/projects/{projectId}/

GET

The VS plugin can issue a GET request to retrieve high level information about a particular project. The {projectId} placeholder in the URL above should be replaced with the ID of the project in question. Here is an example of the output:


```
{
  "project": {
      "id": "iw9w2hlv",
      "name": "Team Project",
      "fullName": "/Project/Team Project",
      "creationDate": "2016-12-03T17:07:27.907-0700"
  },
  "stat": "ok"
}
```


The document includes a “stat” key indicating success and a “project” key with an object of data:

- The “id” key provides the ID of this project
- The “name” key provides the brief name of this project
- The “fullName” key provides the name of this project, including its location within the hierarchy of this personal dashboard.
- The “creationDate” key provides the date when this project was created/launched. This can be useful for identifying newer projects vs older projects.

Project Task List API

http://localhost:2468/api/v1/projects/{projectId}/tasks/

GET

The VS plugin can issue a GET request to retrieve a list of the tasks in a given project. The {projectId} placeholder in the URL above should be replaced with the ID of the project in question. Here is an example of the output:


```
{
  "projectTasks": [
    {
      "id": "3",
      "fullName": "Software Component/Strategy",
      "completionDate": "2017-07-09T15:12:18.919-0600"
    },
    {
      "id": "4",
      "fullName": "Software Component/Code",
      "completionDate": "2017-07-11T11:22:01.615-0600"
    },
    {
      "id": "7",
      "fullName": "Software Component/Test"
    }
  ],
  "forProject": {
    "id": "iw9w2hlv",
    "name": "Team Project",
    "fullName": "/Project/Team Project",
    "creationDate": "2016-12-03T17:07:27.907-0700"
  },
  "stat": "ok"
}
```


The document includes a “stat” key indicating success and a “projectTasks” key with an array of task objects. The tasks appear in chronological order:

- Completed tasks are listed first, in the order they were completed.
- Incomplete tasks are listed next, in the order the individual has planned to work on them.

Each task object includes:

- The “id” key provides the ID of the task
- The “fullName” key provides the name of the task within the given project.
- If the task is finished, the “completionDate” key provides the date when the task was marked complete. This key will be missing for incomplete tasks.

Unlike other JSON documents, the task objects in this array do not have a “project” key. That is because they would all have the exact same value for project, leading to a high amount of redundancy. Instead, the project object is listed once in the top-level document, under the “forProject” key.

Event Listener API

http://localhost:2468/api/v1/events/

GET

During normal operations, certain events can occur within the Process Dashboard application that the VS plugin would like to know about. The VS plugin can issue a GET request to this API to retrieve a list of those events. Here is an example of the output:


```
{
  "events": [
    {
      "id": 150965354,
      "type": "timer"
    },
    {
      "id": 150965355,
      "type": "taskData",
      "name": "completionDate",
      "task": {
        "id": "40",
        "fullName": "Component B/Code Inspect",
        "project": {
          "id": "j5ztgy80",
          "name": "Team Project",
          "fullName": "/Project/Team Project",
          "creationDate": "2017-08-05T14:34:05.376-0700"
        },
        "completionDate": "2017-11-02T13:12:14.000-0700",
        "actualTime": 0
      }
    },
    {
      "id": 150965356,
      "type": "hierarchy"
    },
    {
      "id": 150965357,
      "type": "taskList",
      "name": "Team Project"
    }
  ],
  "nextUri": "/events/?after=150965357",
  "stat": "ok"
}
```


The document includes a “stat” key indicating success. It also includes an “events” key with a list of event notifications. Finally, it includes a “nextUri” key with a URI that can be used to retrieve the next batch of events.

This API is designed to be used as part of a long-polling strategy:

- If any events are waiting in the queue when this API is called, the API will return immediately with a list of those events.
- Otherwise, this API will “hang” for some period of time, waiting for events to occur. Clients of this API should anticipate this potentially long response delay.
- If any events occur during the waiting period, this API will stop waiting and immediately return a list of those events.
- If no events arrive during the waiting period, this API will “time out” and return a document with an empty event list.

When the client receives a response from this API, it should process the list of events (which might be empty). Then it should use the value in the “nextUri” key to immediately perform another request. This pattern should be performed in an endless loop.

This loop should include error handling: if a call to this API fails and a retry doesn’t fix the problem, we can assume that the Process Dashboard has shut down. The loop can stop and the client should visually indicate the broken connection. When the Process Dashboard is restarted and the client reestablishes a connection, the loop should be started back up again.

This API can accept optional query parameters, as shown below:

Parameter Key	Value
`after`	The id of the last event the VS client has handled. The API will only return events with a higher "id" number, possibly waiting until such events become available. (This parameter is used by the "nextUri" key to suggest a URI that will retrieve the next batch of events.)
`maxWait`	The number of seconds to wait before returning an empty document. By default, this is 60 (one minute). Returning an empty document (rather than waiting forever) helps to avoid triggering a time-out error in the VS client's HTTP transport layer.

The events returned by the API will always have a unique (and monotonically increasing) integer ID. Every event also has a type, as described in the table below:

Event Type	Description
`timer`	The timer has been started or stopped. A "timing" field will indicate the new state of the timer.
`activeTask`	The active task has changed. A "task" field will indicate the new task.
`taskData`	Data associated with some task has changed. This could occur if a task has been marked complete/incomplete, if time has been logged against the task, or if its estimated time has changed. The "name" field of the event will provide the name of the attribute that changed, and the "task" field will provide full information about the task that changed. This event will be delivered for changes to any task in the dashboard (not just the active task). If the task represents the active task, the VS client can use the information in this event to update controls like the "Completed" button.
`hierarchy`	A project or task has been created, deleted, or renamed. The client should reload the list of projects and the list of tasks in the current project.
`taskList`	The list of tasks in some project has been changed (for example, the tasks have been reordered). The client should reload the list of tasks in the current project to make sure it has the correct task order.
`notification`	A user notification has been added or removed. The client should call the Notifications API to query the current list of notifications.

More event types will be added in the future to describe other events of interest.

The most important use of this API will be to detect changes that were made in the Process Dashboard by external logic, allowing the VS client stay in sync. For example, if the user opens the main Process Dashboard window and starts the timer there, the VS toolbar can update itself accordingly. But it is worth noting that this API will also send notifications for events that the VS toolbar initiated. For example, if VS uses the Timer API to start or stop the timer, an event will be delivered through this API all the same.

APIs to Show Various Windows

The Process Dashboard uses a task-driven-window design, where different windows are opened to perform various tasks. These task-driven windows can be reused as-is by the VS plugin. To accomplish this, the VS plugin can use the following APIs to open various windows:

- http://localhost:2468/control/showMainWindow - Opens the main toolbar window of the Process Dashboard
- http://localhost:2468/control/hideMainWindow - Hides the main toolbar window of the Process Dashboard, minimizing it to the system tray. (Note: this particular API omits the "window" field from the response, returning only a "stat" field.)
- http://localhost:2468/control/showFindTaskWindow - Opens a window allowing the user to search for (and select) a new active task
- http://localhost:2468/control/showTimeLog - Opens a view of the time log, showing all of the time log entries the user has logged in the past
- http://localhost:2468/control/showDefectLog - Opens a view of the defect log, showing all of the defects the user has logged in the past
- http://localhost:2468/control/showDefectDialog - Opens a small window that allows to user to log a new defect against the active task. (Note: if the active task does not allow the logging of defects, this call will return an HTTP error instead of a response.)

POST

When a POST invocation is made to one of these APIs, the dashboard will display the requested window, and respond with information about the window that was opened. Here is an example of the output:

```
{
  "window": {
       "id": 618306,
       "pid": 1234,
       "title": "Process Dashboard"
  },
  "stat": "ok"
}
```

Due to constraints imposed by the Windows operating system, the newly opened window is very likely to appear behind the VS IDE, and will not automatically receive keyboard focus. To improve usability, the VS code should read the window ID from this document, and use it in a call to the C SetForegroundWindow method.

If no “id” field is present, the VS plugin should next look for a “pid” field. If that field is present and nonzero, the VS plugin should make a call to the C AllowSetForegroundWindow with the value given in that field.

If the document does not contain an “id” field or a “pid” field, the VS plugin should use the value from the “title” field in a call to the C FindWindow method. That method will return an HWND value that should be used in a call to SetForegroundWindow. But this approach should only be used as a last resort when no “id” or “pid” field is present in the response.