XSEDE Enabling Function Use Cases

August 13, 2019

Version 2.0

These are the "enabling functions" provided by the XSEDE system architecture. They are the common ingredients that enable XSEDE to satisfy the use cases in every other category. Each public research computing community has its own enabling use cases: some that overlap with XSEDE's, others that are distinct characteristics of the community.

The purpose of enabling functions is to provide a common set of system-wide features that can be used when satisfying other use cases. Enabling functions arise from reviewing a large set of use cases from different types of community members and identifying the common features they share. For example, most use cases—whether from High-Performance Computing, Data Management, Campus Bridging, or Science Gateways—assume there's a community-wide "login" feature that allows individuals to register, obtain credentials, and then use those credentials to login. Rather than design a custom login feature for every use case, we instead define an enabling function and assume most, if not all, use cases will use it.

XSEDE maintains descriptions of our enabling functions for several reasons. First, they support the introduction of **new services from community members and partners** by identifying features that must be carefully integrated in order to provide a smooth experience for XSEDE community members. Second, these descriptions allow XSEDE to consider **new and better ways** to provide each enabling function without unintentionally losing important features in the process. Finally, they support **partnership and transition discussions** by providing a basis for comparison with other systems.

Each of these enabling functions, with the exception of CAN-05, was originally written as an individual document during the XSEDE-1 project between 2012 and 2015 using a more technical format and terminology. These previous versions are linked in the References section.

CAN-01: Run a remote job

A **science gateway developer** or **application developer** (hereafter referred to as a **"developer"**) needs to enable an application to submit and manage jobs on remote high-performance computing (HPC) or high-throughput computing (HTC) resources. Application examples include: a web application (e.g., science gateway), a workflow system (e.g., Galaxy, Pegasus, Kepler), a campus job queueing service, or a high-throughput system (Condor glide-ins).

We assume the application specifies the computing resource for each job. We also assume that any input data and code needed by the job is already on the computing resource and all results will be left on the computing resource. (Use cases CAN-02 and CAN-03 describe mechanisms for staging data in and out and accessing data remotely.) Finally, we assume that the HPC or HTC resource has its own job queueing, scheduling, and execution mechanisms available for authorized use.

In most cases, the developer wants to experience it as follows.

1. The developer visits the community website, searches for information on remote job submission methods, and uses the information found to develop an application that uses one of the recommended job submission interfaces.
2. When the application needs to submit a job to a computing resource, it invokes the job submission interface to submit the job and receives either an immediate failure response or a success response that includes a job identifier.
3. The application uses the job submission interface to periodically check ("poll") the status of the job--using the job identifier from Step 2--until the status indicates the job has completed. At this point, the application takes whatever action its logic dictates.

The experience will always be as described above unless one or more of the following are true instead.

1. Instead of polling the job submission interface as described in Step 3, the application requests notifications for status changes when it submits the job in Step 2. In this case, the application receives a notification whenever the job status changes.
2. In Step 2, the application requires "at most once" semantics. ("At most once" means a job submission will be executed at most once, even if the application submits it more than once.) In this case, the application provides a unique job identifier for each submission, and the job

submission interface defines a maximum time period within which the "at most once" guarantee is valid.

We'll accept any solution as long as the following are true.

1. In Step 1, the description of each recommended job submission interface includes information about the interface's performance attributes (availability, reliability, scalability) and consistency (for the submission interface itself and for response codes and job statuses) across the various computing resources with which it can be used.
2. In Step 2, the job submission interface must provide a securely authenticated identity for the application (see use case CAN-06) that the resource can use to make authorization decisions and to report usage.
3. In Step 2, if the job submission cannot be accepted by the computing resource for any reason, the job submission interface returns an appropriate failure response.
4. In Step 2, each job's status can continue to be checked for at least 24 hours after it completes.
5. In Step 3, if the job fails to execute or complete for any reason, the job status provides sufficient information for the application (or the developer) to understand what happened.

CAN-02: Managed data transfer

A **researcher**, **educator**, or other **community member** (hereafter referred to as the **"researcher"**) needs to transfer of one or more files and/or directories from one location to another. Once the researcher specifies the data to be moved and the new location, the system accomplishes the transfer without further involvement by the researcher.

We assume the researcher is aware of and knows how to access the community's file transfer service or can find it in the community's documentation. We assume the researcher is registered with the community and is able to authenticate with the transfer service.

In most cases, the researcher expects it to work as follows.

1. First, the researcher opens a web browser and navigates to the community's file transfer service.
2. Then, the researcher uses the interface to specify the source data (files and/or directories) and the destination.
3. Then, the researcher initiates the transfer request. The interface confirms that the request was submitted. At this point, the researcher may leave the system unattended (including logging out, disconnecting) and the transfer will be managed by the system.
4. After the transfer request is submitted and before it completes, the researcher may cancel the transfer.
5. After the transfer request is submitted, the researcher may visit the file transfer service and view the status of the transfer.
6. When the transfer request completes (either successfully or due to an unrecoverable issue), the researcher receives a notification.

It will always be like this unless one of the following is true instead.

1. In Step 3, the researcher requests notifications from the file transfer service when the file transfer state changes. In this case, the researcher will receive a notification when the transfer completes or whenever a significant event occurs (a temporary issue or an unrecoverable issue).
2. Instead of Steps 1-3, the researcher may initiate transfers using an application that's integrated with the file transfer service. In that case, the application receives a request identifier for each transfer request it submits and the application can use the request identifier to check the status of the transfer. The researcher may use either the application interface or the file transfer service interface in Steps 4 and 5. Step 6 is unchanged.

We'll accept any solution as long as the following are true.

1. In Step 1, the researcher must authenticate using the community's standard authentication service. (See CAN-06.)
2. In Step 2, the researcher is able to complete additional authentication for the source and/or destination system if required by the system's access policy.
3. If the researcher uses an integrated application as described in the alternate scenario, the researcher authenticates in the application using the community's standard authentication service (see CAN-06) and the application submits transfer requests using the researcher's identity and credentials. If the source or destination access policies require require additional authentication, the application must enable the researcher to perform the required authentication.
4. In Step 2, the researcher's personal computer (desktop, laptop) can be specified as a source or destination. If the researcher needs to install software to enable this, the software is freely available and is as easy to install as any ordinary application. The software is available for Windows, Mac, and popular Linux systems.
5. In Step 2, there is a freely available mechanism allowing campus IT administrators to make campus systems (including small laboratory servers) available as sources and destinations for authorized individuals. The mechanism is available for popular Linux servers.
6. In Step 2, the researcher can specify that file metadata (modification and access timestamps and mode bits) should be copied from source to destination.
7. In Step 2, the researcher can request data confidentiality (data encryption while "in flight").
8. In Step 4, if the researcher cancels a transfer, any data already transferred to the destination remains there. The transfer status should indicate the request was cancelled by the researcher.
9. Step 5 can happen up to one month after a transfer request completes.
10. In Step 5, the transfer status includes: whether the transfer has started, completed, or failed; the number of bytes and files transferred successfully; the total transfer time; and bandwidth utilization data.
11. In Step 6, email notification is available as an option.
12. If the researcher's request cannot be accommodated by the file transfer service (for example, if the researcher specifies a destination to which the researcher doesn't have write access or if there is insufficient storage on the destination), the error message returned is consistent, meaningful, and helpful.
13. Transfers make full use of the connectivity and hardware between source and destination, with respect given to current traffic contention. E.g., if source and destination both have 10 Gbit/s

connections to a common backbone network and both source and destination have reasonably designed data transfer hardware, transfers should regularly achieve 500 Mbit/s - 5 Gbit/sec depending on traffic contention.

14. Transfer performance is not affected by the location, connectivity, or equipment used by the researcher.
15. The file transfer service automatically restarts transfers that are interrupted by transient failures. Examples of transient failures include: network interruption, source or destination system failure or misconfiguration, failure of any intermediate operation, checksum failure, user credential expiration.
16. The file transfer service performs data integrity checks on all transfers. Transfers are retried until the integrity checks succeed, up to a defined maximum number of retries.
17. When a transfer restarts, data that has already been transferred is not transferred again.
18. The file transfer service can maintain at least 1,000 active transfer requests (submitted and not yet completed) at a time.
19. The file transfer service itself can be restarted without losing track of queued, active, or completed file transfer requests.
20. The file transfer service itself is available 99.7% of the time. (No more than 26.28 hours of downtime per year.)

CAN-03: Remote file access

A **researcher** needs to create, read, update, and delete files and directories from one site at another using POSIX operations, e.g., creat, read, write, unlink. Note that "site" can refer to a resource provided by a service provider, a local campus, a research lab, a desktop computer, or even a home computer

We assume the following things.

1. The client is properly authenticated, and has required permissions.
2. The client is familiar with Unix file commands, and applications use the standard POSIX file system commands.
3. The file(s) and directory(s) being accessed exist.
4. There is sufficient space for those file(s) and directory(s) on the destination file system.
5. The file and directory services, source file system, destination file system, and intervening network do not fail during execution of the file transfer.

In most cases, the researcher expects it to work as follows.

1. The researcher issues a POSIX compliant file system call on the remote file system.

It will always work like this except when one of the following is true.

1. The researcher accesses the remote file system or storage using command line tools.
2. The researcher accesses the remote file system or storage using a GUI.
3. The researcher accesses the remote file system or storage using an API other than POSIX.
4. The authorized researcher changes the access control list on a file or directory.

We'll accept any solution as long as the following are true.

1. The file and directory services are available 99.7% of the time. (No more than 26.28 hours of downtime per year.)
2. Unauthorized access of a file or directory results in an error message that is consistent, meaningful, and helpful.
3. Changes to the remote file system are visible to the researcher within 30 seconds.
4. Consistency and update semantics are clearly defined.

CAN-04: Open a command shell on a login server

A **community member** needs to open a command shell on a login server. The login server is provided by a community *service provider*. We assume the community member is registered with the community (and thus has a *community identity* with associated *credentials*), knows the name/address of a login server, and is authorized to use the login server.

In most cases, the community member expects it to work as follows.

1. The community member opens a connection to the login server and logs in.
2. Then, the community member is presented with a command shell and is able to enter commands.

It will always work like this except when the community member also needs to run programs that open graphical interfaces on the community member's local device (desktop, laptop, mobile). In that case, before Step 1, the community member must open or activate a window server (e.g, an X window server) on the local device and the community member must set an environment variable in the command shell that specifies the address of the local device. We also assume connections from the login server are allowed by the window server, the local device, and the network.

We'll accept any solution as long as the following are true.

1. The community member can complete this use case using any device with a standard (up-to-date) web browser. (The alternate scenario also requires a window server.)
2. The community member can complete this use case using any device with a standard (up-to-date) SSH client. (The alternate scenario also requires a window server.)
3. In Step 1, the community member can login using the community member's community identity and credentials.
4. In Step 1, the confidentiality of the community member's credentials is never compromised. (The connection is encrypted and confidential and the community member's credentials are never visible on a display.)
5. In Step 1, if the access policy of the service provider that operates the login server requires it, the community member can provide a second authentication factor.
6. If the command shell session is lost for some reason, the community member can reattach to the session after reconnecting.

CAN-05: Record and summarize the use of a service

A **service provider** (someone who administers or operates a service that is part of a community) needs to report the use of their service to the community's management for use in system reports and

planning processes. We assume the service already has an internal mechanism for logging/recording its use.

In most cases, the **service provider** wants to experience it as follows.

1. First, the provider locates and reviews the documentation for the mechanisms the community provides for reporting use of a system component and viewing component usage summaries.
2. Then, the provider installs the required software on a local system.
3. Then, the provider configures their service and the reporting software as described in the documentation.
4. Then, the provider waits for an appropriate duration (specific to the selected mechanism) and confirms through the community's usage summary views that their usage data has been collected and summarized. The operator files support tickets if there are issues with the reporting mechanism or if their usage is missing in summary views.

We'll take any solution, as long as the following are true.

1. The solution should make use of the service's current logging/usage reporting mechanisms, instead of requiring the service's developers to support an additional mechanism.
2. The solution should honor service provider policies regarding release of personally identifiable information (PII) to partners.
3. The solution should support whatever degree of detail (in the usage data) is specified in agreements between the service provider and the community.
4. The documentation for service providers notes that service providers are responsible for usage data they release to the community, and their users should be notified of the practice in their end-user privacy policy or terms of service.
5. Usage data collected by the community—including both detailed and summary data—are only accessible by authorized entities as specified by the community, service providers, and applicable laws.

CAN-06: Authenticate with an application

A **community member** needs to securely share his or her identity with an application in order to use a feature that requires authorization. The community member is a researcher, student, staff member, or other involved party. The application may be a website or web application, a locally-installed application or mobile app, a network service (e.g., SSH, GridFTP), or an application interface (API) accessed via software.

We assume the community member is already registered with an identity provider (IDP) and can authenticate with it. We also assume the application is configured to use an authentication service associated with the IDP.

In most cases, the community member wants to experience this as follows.

1. The community member interacts with the application and requests something requiring authentication. (Some applications may require authentication immediately when the

application is accessed. Others may only require authentication for specific features or to access specific data.)

2. The application engages the authentication service, providing details about the credentials necessary to enable the feature(s) the community member requested.
3. The authentication service either enables the individual to select an identity provider (IDP) for authentication or it selects one based on the credentials that were requested.
4. The community member authenticates with the IDP.
5. The authentication service returns one or more credentials to the application based on the credentials that were requested and the identity returned by the IDP.
6. (Optional) Subject to permission granted by the community member, the authentication mechanism also returns delegated credentials that enable the application to act on the individual's behalf with other services.
7. The application acknowledges that the community member is authenticated and uses the credentials to authorize (or not, as appropriate) specific features.

It will always be like this except when the following is true.

1. In Steps 2-5, the community member may already possess a secure credential issued by an IDP (e.g., an X.509 certificate or proxy certificate) and may wish to use that instead of an authentication service. In this case, the application must be on the same system as the credential. The application should validate the credential using trust chain validation and a store of trusted root certificates, obtaining the community member's identity from the credential. It can use the credential to authenticate cryptographically with remote services on the community member's behalf. In Step 6, remote services may optionally receive delegated credentials via the authentication process, enabling them to interact with further services on behalf of the community member.

We'll accept any solution as long as the following are true.

1. In Step 3, the authentication service enables use of a wide variety of IDPs, including those used by academic and research institutions (InCommon, eduGAIN), OSG, PRACE, and EGI.
2. In Step 4, the community member's interaction with the IDP must be confidential, even from the application.
3. The credentials given to the application may only be delegated credentials, meaning they're specific to the application, can only be used for a limited set of actions, and can be revoked by the community member.
4. Applications must be able to add and delete credentials from active sessions.
5. Information about failures should be informative and useful to both community members and administrators.
6. All communication involving identity information or credentials must be reliable and confidential using best practices and techniques.

CAN-07: Subscribe to resource status information

An **application developer** or **science gateway developer** needs to create an application that subscribes to resource status information so the application receives asynchronous, best-effort updates when the

status changes or is refreshed. Application examples include: a web application (e.g., science gateway or user portal), a workflow system (e.g., Galaxy, Pegasus, Kepler), a mobile application, or a system monitoring service.

We assume the developer is aware of the information system and its documentation. We assume the developer is registered with the community and has credentials for authentication.

In most cases, the developer wants it to work as follows.

1. First, the developer reads the documentation for the information system, finds the material on subscribing for updates, and uses the material to develop an application that uses the subscription interface.
2. Then, the developer registers the application with the community, obtains any credentials needed to allow the application to use the information service, configures the application, and deploys it for use.
3. When the application needs information updates, it creates a subscription with the information system, specifying: the information of interest, how the information should be delivered, whether or not buffering is requested when the application isn't ready to receive information (including any buffering limits), and a maximum time period for the subscription.
4. While the application's subscription is active, whenever the information system has information of interest to the application:
    a. If the application is ready to receive, the information system delivers the information to the application.
    b. If the application isn't ready to receive and has requested buffering, and the information system supports buffering, the information system buffers the resource information for later delivery.
5. While the application's subscription is active, whenever the application becomes ready for delivery of information:
    a. The application informs the information system that it is ready.
    b. If the information system supports buffering, it delivers any information that has been buffered for the application.
6. While the application's subscription is active, whenever the application is no longer ready to receive information, the application informs the information system that it is no longer ready to receive information.
7. The subscription expires when the subscription times out or when the application deletes the subscription.

We'll accept any solution as long as the following are true.

1. In Step 1, documentation is comprehensive and easy-to-follow.
2. In Step 1, any required software (e.g., SDK) is easy to install and configure. Community support for the required software is available.
3. In Step 3, the application authenticates with the information service as described in use case CAN-06. Authentication may use the application's credentials (obtained by the developer and configured during deployment) or the application user's credentials (obtained when the user logs into the application).

4. In Steps 3-5, access to the subscription function and access to the information are both based on the identity returned from authentication.
5. The subscription feature supports both data integrity and data confidentiality via encrypted communication.
6. Any community policies regarding the use and sharing of information are readily available to developers.
7. The information system can deliver at least ten messages per second to a single application.
8. The information system can deliver, in aggregate, at least 150 messages per second and 3 MB/sec to all current subscriptions.
9. The information system is available 99.9% of the time. (No more than 8.77 hours of downtime per year.)
10. When the information service becomes aware of updated data, it takes no more than one second for an update to be delivered to the relevant subscribers.
11. When the information system buffers data for a subscription, the data will remain available per the subscription's buffering rules (e.g. maximum buffer length or maximum buffer duration) unless it is explicitly deleted.

CAN-08: Search for resource information

A **community member** needs to request resource information of a particular type and receive relevant results. We assume the community member is aware of the search system and its interface. We assume the community member is registered with the community and has credentials for authentication.

In most cases, the community member wants it to work as follows.

1. First, the community member opens a web browser, navigates to the search interface, and enters a query specifying the information of interest.
2. Then, the interface displays zero or more search results.

It will always work like this unless the following are true instead.

1. In Step 1, the community member uses a locally-installed application (instead of a web interface) to perform the search. In this case, the results in Step 2 are displayed by the application.
2. The community member needs to develop an application that can perform searches. In this case, the community member first reads the documentation for the community's information system, finds the material on the search interface, and uses the material to develop an application that uses the search interface. Then, the developer registers the application with the community, obtains any credentials needed to allow the application to use the search interface, configures the application, and deploys it for use. In Steps 1 and 2, the application replaces the search interface.
3. In Step 2, the search results may be returned in pages or chunks instead of all at once. In this case, the interface provides a way to access each page or chunk.

We'll accept any solution as long as the following are true.

1. In Step 1, the community member is able to authenticate with the search interface as described in use case CAN-06.
2. In Step 2, the search results can be filtered based on the authenticated identity (or absence of an authenticated identity).
3. In the scenario where the community member is developing an application, the application is able to authenticate with the search interface as described in use case CAN-06 either using its own credentials (obtained by the developer and deployed with the application) or the application user's credentials (obtained when the user logs in to the application).
4. In the scenario where the community member is developing an application, documentation is comprehensive and easy-to-follow.
5. In the scenario where the community member is developing an application, any required software (e.g., SDK) is easy to install and configure. Community support for the required software is available.

CAN-09: Establish and manage a community identity

A **community member** needs to to establish, manage, and disable a community identity. We assume the community member is aware of the existence of the community, has access to the Internet, and has a standard web browser.

In most cases, the community member wants to experience it as follows.

1. The community member visits the community's website and registers, creating a community identity.
2. The community member manages his/her profile. This can be done at any time, and repeatedly.
    a. (optional) The community member may use an identity from another community (e.g., academic institution) to pre-populate the profile, as opposed to starting from scratch. (Think: "Sign up using Facebook.")
    b. The community member binds one or more email addresses with his/her profile. An email address is validated by sending an email to that address with a link that the community member clicks on to validate the email address. A single email address can only be bound to one community identity. For each community identity, one email address is designated the "primary" email address.
    c. (optional) The community member binds (federates) one or more identities from other organizations (e.g., InCommon) with his/her community identity. An external identity can only be bound to a single community identity.
    d. (optional) The community member adds, modifies, deletes, and controls visibility of other attributes in the profile.
    e. (optional) The community member resets the password on his/her community identity via an email address in his/her profile, or based on information in the profile.
3. The community member logs into any community service as described in use case CAN-06.
4. (optional) A project in which the community member is a member receives an allocation and the community member uses his/her community identity to access the associated resource.
5. (optional) The community member manages groups.

a. The community member creates any number of groups, automatically taking on an administrator and manager roles within these groups.
b. The community member configures the profile for a group for which he/she is an administrator (e.g., description, group visibility, member visibility, membership admission policy).
c. The community member adjusts the membership of a group for which he/she is a manager, by adding or removing community identities to/from the groups, either individually or in bulk, and specifying roles for each identity (e.g., member, manager, administrator).
d. The community member invites other community members to join a group for which he/she is a manager by sending invitations to their community identities or email addresses.
e. The community member requests membership in a group managed by another community member.
f. The community member disables a group for which he/she is an administrator. A disabled group is no longer usable for authorization actions.
g. The community member enables a previously disabled group for which he/she is an administrator.
h. The community member deletes groups for which he/she is an administrator.

6. (optional) Groups can appear in authorization policies controlling access to a resource (e.g., file or storage system or use of an allocation). The authorization policies are determined by the parties responsible for the services/resources in question.
7. The community member discovers a list of groups in which he/she has a specific role.
8. The community member discovers a group's profile.
9. A community identity can be disabled so that it can no longer be accessed.
    a. A community member can disable his/her community identity.
    b. Administrators with appropriate rights can disable/re-enable a community identity.

We'll accept any solution as long as the following are true.

1. A community member who doesn't currently have a community identity can establish one in five minutes or less, without direct synchronous interaction with another human being.
2. A community member can bind an identity from another organization (e.g., InCommon) to his/her community identity in five minutes or less, without direct synchronous interaction with another human being.
3. The community's identity management interface is fully accessible via commodity web browsers and Internet connections.
4. The community's authentication system is mission critical and must be available 99.95% of the time. (No more than 4.38 hours of downtime per year.)
5. The identity update/management interface is available 99.9% of the time. (No more than 8.77 hours of downtime per year.)
6. The community member can adjust any editable fields in his/her profile with no direct interaction with another human being.

7. The community member can reset his/her community identity's password via email in five minutes or less. (We assume the community member can access the registered email address.) A mechanism must exist that allows the community member to accomplish this task without direct synchronous interaction with a human.
8. The community member can complete individual group management tasks in five minutes or less and without direct synchronous interaction with another human being.
9. Groups can have at least several thousand members.
10. Group membership tests should take less than five seconds.
11. Changes to group membership will take effect in the identity management system in less than five seconds. Group membership changes may not be visible to an application until an affected member re-authenticates.
12. Applications can use community group functions without direct synchronous interaction with community administrators.

CAN-11: Publish resource status updates

An **application developer** or **science gateway developer** needs to create an application that publishes resource status updates in a best-effort, asynchronous manner. Resource examples include: a computing resource (e.g., a high-performance or high-throughput computing resource), a monitoring service, a scientific instrument (e.g., a sensor network, telescope, microscope), a running simulation, or a web application (e.g., science gateway or user portal). Status information may include: availability information (up/down), service information (active queues, queue length, estimated queue times), progress toward completion (units completed, percent completion).

We assume the following things.

1. The developer is aware of the information system and its documentation.
2. The developer is registered with the community and has credentials for authentication.
3. The developer already has a mechanism for detecting/gathering the resource's status.
4. The developer is responsible for the quality of any published information. (The information is accurate, up-to-date, and conforms to appropriate schemas.)
5. The developer includes appropriate timestamps and validity time ("time to live" or TTL) in any information they publish.

In most cases, the developer wants it to work as follows.

1. First, the developer reads the documentation for the information system, finds the material on publishing status updates, and uses the material to develop an application that uses the publishing interface.
2. Then, the developer registers the application with the community, obtains any credentials needed to allow the application to use the information service, configures the application, and deploys it for use.
3. When the application detects a change in resource status or needs to refresh the status information, it publishes the status update, supplying any required metadata (topic, queue, channel, acknowledgement request).

4. If the application requested acknowledgement in Step 3, and if the information system supports acknowledgements, the information system acknowledges acceptance of the information.

We'll accept any solution as long as the following are true.

1. In Step 1, documentation is comprehensive and easy-to-follow.
2. In Step 1, any required software (e.g., SDK) is easy to install and configure. Community support for the required software is available.
3. In Step 3, the application authenticates with the information service as described in use case CAN-06. Authentication uses the application's credentials (obtained by the developer and configured during deployment).
4. In Steps 3-4, access to the publishing function (permission to publish updates with specific metadata) is based on the identity returned from authentication.
5. The publishing function supports both data integrity and data confidentiality via encrypted communication.
6. Any community policies regarding information sharing (e.g., handling of sensitive data and/or personally identifiable information, privacy) are readily available to developers.
7. The information system only transports what the application has produced.
8. In Steps 3 and 4, when the application requests an acknowledgement, the amount of time between sending the update request and receiving the acknowledgement is less than 1 second. (This does not include the time to propagate the information to any subscribers.)
9. The application can publish at least two updates per second without data loss.
10. The information service can accept, in aggregate, up to 150 updates/sec and 3 MB/sec from all publishers.
11. The publishing function is available 99.9% of the time. (No more than 8.77 hours of downtime per year.)
12. When an update is accepted, it remains available in the system until it's validity time (TTL) expires or until it is explicitly deleted.

CAN-12: Update resource information

A **community member** needs to add or update resource information. "Resource information" includes the description and details about a community resource. Examples of resources include: a computing resource (e.g., a high-performance or high-throughput computing resource), a web application (e.g., science gateway, user portal, online training site), a service (e.g., data transfer service, storage service, web API).

We assume the following are true.

1. The community member is aware of the information system and its interface.
2. The community member is registered with the community, has credentials for authentication, and is authorized to update information for a resource.
3. The community member is responsible for the quality of the information provided. (The information is accurate and up-to-date.)

4. The community member supplies an appropriate validity period (“time to live” or TTL) with any edited information as prompted by the interface.

In most cases, the community member wants it to work as follows.

1. The community member opens a web browser, navigates to the information interface for the resource, and clicks “Edit.” (The exact wording in the interface may vary.)
2. The community member edits the resource’s information and clicks “Submit.” (The exact wording in the interface may vary.)
3. The interface acknowledges that the edits have been accepted.

It will always work like this unless one of the following is true instead.

1. In Steps 1, the community member uses a locally-installed application (instead of a web interface). In this case, the application provides the interfaces for Steps 1-3.

We’ll accept any solution as long as the following are true.

1. In Step 1, if the community member hasn’t already authenticated before clicking “Edit,” the community member must authenticate as described in use case CAN-06 before proceeding.
2. In Step 2, access to the editing function (permission to edit specific resource information) is restricted to authorized individuals for each resource.
3. The editing function supports both data integrity and data confidentiality via encrypted communication.
4. Any community policies regarding information sharing (e.g., handling of sensitive data and/or personally identifiable information, privacy) are readily available to community members.
5. The information service can handle an aggregate of at least 100 million entries and 100 GB of data.
6. The editing function is available 99.9% of the time. (No more than 8.77 hours of downtime per year.)
7. When an edited entry is accepted, it remains available in the system until it’s validity time (TTL) expires or it is explicitly deleted.

References

[1] **XSEDE Canonical Use Case 1: Run a Remote Job.** F Bachman, J Brown, I Foster, A Grimshaw, A Hossain, D Lifka, M Riedel, S Tuecke. Aug 28, 2013. (http://hdl.handle.net/2142/45685)

[2] **XSEDE Canonical Use Case 2.0: Managed File Transfer.** F Bachman, J Brown, I Foster, A Grimshaw, A Hossain, D Lifka, L Liming, M Riedel, S Tuecke. Mar 20, 2014. (http://hdl.handle.net/2142/48673)

[3] **XSEDE Canonical Use Case 3: Remote File Access.** F Bachman, J Brown, I Foster, A Grimshaw, A Hossain, D Lifka, M Riedel, S Tuecke. Aug 28, 2013. (http://hdl.handle.net/2142/45687)

[4] **XSEDE Canonical Use Case 4: Interactive Login.** F Bachman, I Foster, A Grimshaw, A Hossain, D Lifka, M Riedel, S Tuecke. Dec 11, 2013. (http://hdl.handle.net/2142/46550)

[5] **XSEDE Canonical Use Case 6: Authenticate to one or more SP resources, SP services, and XSEDE central services.** Jan 20, 2015. (http://hdl.handle.net/2142/88830)

[6] **XSEDE Canonical Use Case 7: Subscribe for Resource Information.** F Bachman, J Brown, I Foster, A Grimshaw, A Hossain, D Lifka, JP Navarro, M Riedel, W Smith, S Tuecke. Mar 27, 2014. (http://hdl.handle.net/2142/48913)

[7] **XSEDE Canonical Use Case 8: Search for Research Information.** F Bachman, J Brown, I Foster, A Grimshaw, A Hossain, D Lifka, JP Navarro, M Riedel, W Smith, S Tuecke. Mar 27, 2014. (http://hdl.handle.net/2142/48903)

[8] **XSEDE Canonical Use Case 9: XSEDE User Identity and Access Management.** I Foster, A Grimshaw, A Hossain, D Lifka, M Riedel, S Tuecke. Jul 30, 2013. (http://hdl.handle.net/2142/45237)

[9] **XSEDE Canonical Use Case 11: Publish Resource Information.** F Bachman, J Brown, I Foster, A Grimshaw, A Hossain, D Lifka, JP Navarro, M Riedel, W Smith, S Tuecke. Mar 27, 2014. (http://hdl.handle.net/2142/48904)

[10] **XSEDE Canonical Use Case 12: Update Resource Information.** F Bachman, J Brown, I Foster, A Grimshaw, A Hossain, D Lifka, JP Navarro, M Riedel, W Smith, S Tuecke. March 27, 2014. (http://hdl.handle.net/2142/48905)

History

	Version	Date	Changes	Author
Entire Document	2.0	8/13/2019	Collated all of the enabling function use cases (01-12) for the first time, reformatted all use cases (with the exception of CAN-05) from the XSEDE-1 format to the simpler XSEDE-2 format, and removed unnecessary XSEDE terminology.	L. Liming